2005

Securing Peer-to-Peer Overlay Networks

William Heinbockel

Thesis/Dissertation Author Permission Statement

Title of thesis or dissertation: SECURING PEER-TO-PEER OVERLAY NETWORKS

Name of author: WILLIAM HEINBOCKEL
Degree: MS COMPUTER SCIENCE
Program: VLSGG
College: GOLISANO GCCIS

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:

I, William Heinbockel, hereby **grant permission** to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author: William Heinbockel Date: _____

Print Reproduction Permission Denied:

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Inclusion in the RIT Digital Media Library Electronic Thesis & Dissertation (ETD) Archive

I, William Heinbockel, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity.

I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for ETDs.

I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author: William Heinbockel Date: _____

Abstract

Overlay networks are virtual networks, which exist on top of the current Internet architecture, and are used in support of peer-to-peer (P2P) applications. The virtualization provides overlays with the ability to create large, scalable, decentralized networks with efficient routing. Many implementations of overlay networks have come out of academic research. Each provides a unique structure and routing configuration, aimed at increasing the overall network efficiency for a particular application. However, they are all threatened by a similar set of severe vulnerabilities.

I explore some of these security deficiencies of overlay network designs and propose a new overlay network security framework — Phyllo. This framework aims to mitigate all of the targeted security problems across a majority of the current overlay implementations, while only requiring minimal design changes. In order to demonstrate the validity of Phyllo, it was implemented on top of the Pastry overlay architecture. The performance and security metrics of the network with the proposed framework are evaluated against those of the original in order to demonstrate the feasibility of Phyllo.

Acknowledgements

I would like to extend my gratitude towards my advisor, without him Phyllo would have never seen the light of day. Also, he pushed me to submit a more compact version of this project to the NPSec Conference, at which we will be presenting in November 2005.

I would also like to thank the other members of my board for putting up with my quarter of absence from this thesis to relocate to the Boston, MA area. Additionally, thank you to Jim Moore for getting me started in the right direction in order to break into the field of InfoSec. The MITRE Corporation deserves acknowledgment for providing me an awesome job, as well as my coworkers who encouraged me to complete this paper.

What's a man to do without his friends? I know that my friends provided me with an enjoyable college experience. Their much appreciated distractions kept me from getting too entwined in this thesis venture and reminded me that "all work and no play makes Jack a dull boy"

Last, but definitely not least, I have to give praise to my parents. Without their love and encouragement, I would not be where I am today. They have always been there to provide emotional support, especially during the worst of times.

Forward

A shortened version of this thesis has been submitted to, and accepted for publication at the First Workshop on Secure Network Protocols (NPSec). It will be held in conjunction with the IEEE ICNP Conference in Boston, MA in November 2005 [1].

Contents

List of Figures

Chapter 1

Introduction

> There was of course no way of knowing whether you were being watched at any given moment. How often, or on what system, the Thought Police plugged in on any individual wire was guesswork.
>
> — George Orwell, *1984*[1]

> There's a war out there, old friend. A world war. And it's not about who's got the most bullets. It's about who controls the information. What we see and hear, how we work, what we think... it's all about the information!
>
> — Cosmo, *Sneakers*[2]

What if someone could exert complete control over your life — what you could say, who you could interact with, where you could go...? For example in George Orwell's *1984*, he describes a futuristic setting, wherein every citizen's actions and were monitored by the government. Anybody that was deemed to be a threat was eliminate. Even if they thought about something that they should not, the Thought Police would erase them from existence.

[1] G. Orwell, "1984," New York: Signet, 1992.

[2] "Sneakers," Dir. P.A. Robinson, Perfs. R. Redford, D. Aykroyd, R. Phoenix, DVD, Universal Pictures, 1992.

Nobody in their right mind would ever willfully allow such Orwellian dominance. However, this is exactly what a terrorist can do over the Internet, and no one seems to mind.

Through security flaws in the architectural design of peer-to-peer (P2P) overlay networks, attackers can control information routes. These networks, which can be used for applications ranging from file sharing to enabling battlefield communications, could end up being controlled by the enemy. Why should we just give them the upper-hand in this "information war"?

In order to fight back and mitigate these flaws, I propose a security framework, Phyllo, that can be implemented on any overlay network. This framework acts to reorganize the network into groups, allowing for the untrusted members to be separated from the entire network. Once separated, barriers can be set up to protect against any attackers and monitoring systems put in place to help identify the trustworthy in a sea of unknowns.

In an attempt to understand this P2P security framework, the reader first must be familiarized with the concept behind P2P applications and their overlay networks. Chapter 2 provides an introduction to these technologies as well as an insight into their history (Section 2.2). Chapter 3 examines the security issues brought about through the architectural design of overlay networks and discusses the motivations behind Phyllo.

From there, we begin to dig into the goals and details of the Phyllo framework. In Chapter 4, an overview is provided into the modifications and security advantages of the proposed framework. This establishes the introduction necessary to describe actual implementation details of applying Phyllo to Pastry [2], an overlay network substrate developed out of Rice University. To demonstrate the validity and minimal overhead of Phyllo, results are given and discussed (Chapter 6). Finally, this thesis

concluded with a look at some previous projects similar to Phyllo (Section 2.4) and various possible routes for future work with Phyllo in Section 7.1.

Chapter 2

Overlay Networks

> P2P its a revolutionary technology. P2P is technically unstoppable.
>
> Most of all, P2P its positive for companies, for the market and its good for users.
>
> —Marco Montemagno, *P2P Manifesto*[1]

Phyllo is a security framework for overlay networks. Now, what is meant by the term "overlay network"?

2.1 Background

Many people in today's Information Age are familiar with the Internet. They may even know that the Internet is comprised of many interconnected computer systems, or networks. Therefore, it must follow that an overlay network is also a group of interconnected computers, right? Well, yes and no.

Computers that are part of an overlay network are indeed connected to one another. However, unlike other networks, this connection is not a concrete concept. With the Internet, two computers communicate by establishing a more-or-less direct

[1] M. Montemagno. *P2P Manifesto.* 2005. Available Online: `http://rothko.hmdnsgroup.com/~montemag/p2pmanifesto_en.pdf`

connection to each other. Overlay networks use a more abstract connection scheme. Instead of sending a message directly to its destination, overlays forward messages towards a destination. The distinction here is subtle, but very important; perhaps an illustration would make these concepts more clear.

Assume that you had an important message to deliver to a friend. If you and your friend were members of a typical network (e.g., the Internet), you would deliver the message directly. This could be accomplished by either going out and meeting your friend personally, or calling your friend and speaking to him/her directly. In both instances you are the one delivering the message, and you know that the person that you are talking to is indeed your friend.

Similarly, if you had the same important message that needed to be delivered to the same friend, but in this case both of you are members of an overlay network. You might decide to use a courier service to deliver the message. With overlay networks, there is the introduction of some third-party who you must depend on to deliver the message. You must trust that this courier: 1) deliver the message and 2) that the message delivered was the correct message (i.e., the message was not modified in transit). One of the most famous examples of such indirect communication is the Byzantine Generals Problem, which will be discussed later.

It is important to note that this type of overlay network communication is often referred to as peer-to-peer communication.

2.1.1 Peer-to-Peer v. Overlay

Though the terms "peer-to-peer" and "overlay" are often used interchangeably with regards to computer networks, they are actually two different concepts. An overlay network, as detailed previously, is a specific network architecture; it describes how

the nodes — the computer systems that comprise the network — are arranged and how they communicate. Here, the adjective "overlay" means that this network actually exists on top of an existing network structure. It is often thought of a virtual network that acts as a parasite, leeching off of another network's (e.g., the Internet) resources. A diagram of the overlay network host network relationship can be seen in Figure 2.1.

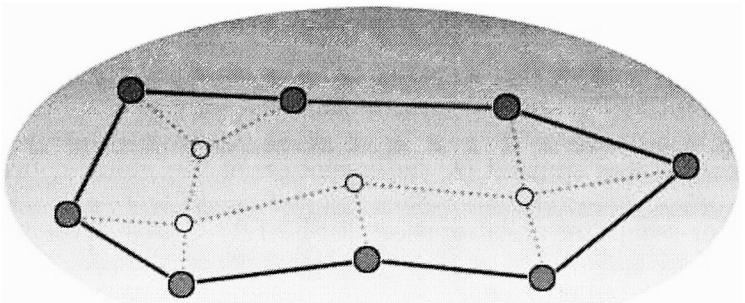

Figure 2.1: Depiction of a Generic Overlay Network. The nodes (gray circles) and their virtual connections (solid, black lines) are shown overlaying routers (white circles) and links (dotted lines) of the base network structure.

In Section 2.1, peer-to-peer communication was introduced to describe the courier-type message transmission employed by overlay networks. To make things even more confusing, there are applications referred to as peer-to-peer (P2P) applications. Some examples of such applications include some famous filesharing programs: Napster, Gnutella, Bit Torrent, etc. The feature that qualifies these applications as P2P is that they use overlay networks to efficiently search for and transfer files. P2P applications provide the logic and information, while the overlay network takes care of routing the data from its source towards its intended destination.

However with that said, both terms will be used liberally throughout the remained of the paper to refer to the combination of P2P applications and their overlay networks. Not much differentiation will be made between the two concepts,

as they are primarily found coupled together. More technical details as to how an overlay network operates, and how it combines with P2P applications will be discussed in Chapter 4.

2.2 History

2.2.1 The Evolution of P2P

As explained in [3], the P2P network was actually a forefather to the modern-day Internet. Its creation was spurred by the invention of the modem, bringing about a demand for inter-computer collaboration. USENET [4] was developed to fill this void. It was a simple bulletin board system, designed by two graduate students at Duke University, where users could dial-in to a main server to read and post messages. This basic design was remarkably similar to that of Napster, which would come a couple of decades later. The problem with USENET was the reliance on a single server. Whenever the traffic spiked or the system went down, the service was unavailable.

Tom Jennings attempted to solve this problem with FidoNet [5]. Though developed independently of USENET, it created a USENET-like bulletin board system over a more decentralized environment; a similar transition to that which would occur 16 years later in 2000, when Gnutella developed off of Napster. Over time, USENET and FidoNet managed to solve many of their problems with scalability and security, so why are we still faced with some of the same issues? As quoted from George Santayana and reiterated in [3], "Those who do not remember the past are condemned to repeat it."

Unfortunately, the later developers of similar networks did not learn from their predecessors. Over the past couple of years, there have been a large number of

P2P networks and applications introduced, all of which have the same fundamental problems. These security problems are still evident in some of today's overlay networks (Chapter 3). Some of these more recent (late 1990's) P2P systems include: Freenet [6], Napster, and Gnutella. Freenet's primary focus was to use the concept of overlay networks to provide anonymous web browsing. However, its achievements were dwarfed by the success of the file sharing applications of the time.

The most popular of these applications was Napster [7]. Napster was developed by Shawn Fanning, in order to share music with his friends. People were able to connect to the network to share their music, search other's music lists for specific songs, and then download them. From there, employees at Nullsoft realized the potential to improve the overlay network to better suit this type of application. This improvement led to the creation of Gnutella [8]. AOL, the parent company of NullSoft, soon terminated its development, calling it an "unauthorized freelance project". Eventually, Gnutella found its way into the public. While Napster concentrated solely on the sharing of digitized music, Gnutella allowed for a broader range of files to be shared. Both quickly became very popular applications until around 2001, when they would be brought to a halt because of copyright lawsuits. Since then, Napster has reemerged as a music subscription service, and the Gnutella protocol has been embraced by many other file-sharing applications looking to piggyback on these applications' fame.

2.2.2 Modern P2P Applications

Several overlay networks have grown out of the post-Napster decentralization push: Chord, CAN, Pastry, Tapestry, and Spread are only a portion of them. These virtual, ad hoc network structures can easily create large, scalable, application-specific networks. Since their resurgence, many useful applications have be found. Exam-

ples include file storage [9], efficient media streaming [10], messaging services [11], and Internet telephony services, such as Skype [12]. With this paper, we will be concentrating on the architectures and protocols of Pastry and Chord.

Pastry [2] was developed out of Rice University in 2001. It was designed exploit the underlying network topology for reduced latency, while using consistent hashing to provide efficient lookup times. Additionally, like all overlays, Pastry scales well and is fully decentralized.

At around the same time, MIT was creating Chord [13]. While fundamentally similar to its Rice University counterpart, Chord does not take advantage of the underlying network. However, this reduces the overall complexity of the system. Does this reduced complexity translate into a more secure system?

2.3 Architecture

The architecture of overlay networks has not changed much from its original beginnings in USENET and FidoNet. New techniques to enhance routing, storage, and searching capabilities within these networks have improved, along with the speed and quality of the underlying network. Each of the important architectures along the evolutionary path of these unique networks will be sufficiently explored to provide a basis for understanding security issues. Special emphasis will be placed on Chord and Pastry.

2.3.1 USENET and FidoNet

USENET and FidoNet marked the beginning of peer-to-peer interaction, by way of computer networking. With each of these systems, each participating computer on the network, a node, was assigned a unique identification number, called the nodeId.

In an attempt to minimize the number of long-distance phone calls, FidoNet implemented the identifier, `zone:net/node`, to label nodes. The idea was to have systems interact by calling the other nodes that were geographically closest to them. The `zone` would identify a large area (e.g. a continent) where the node was contained. From there, the identity became more and more specific, breaking each `zone` into `nets`, and finally to the actual `node`. With this structure, the information could be slowly spread across the globe. Additionally, the system was configured to have the modems network at night, when the phone rates were the least expensive.

Since such networks overlaid the phone grid, they were faced with a slightly different set of issues. Firstly, the network only existed for a short period of time. This means that an potential attacker had only a brief window to damage the network. Also, efficiency was not a real issue. If the network failed one day, they would just try again on the next. Secondly, new members had to be manually added to the network's dial-up list. Only real life acquaintances of current members were usually permitted to become hubs. The issue of trusting anonymous or unknown users was nonexistent. Even though many of today's security problems were present in USENET and FidoNet, other factors from the technology to the implementation shadowed such deficiencies.

2.3.2 Napster and Gnutella

Despite the lapse of more than a decade from the birth of USENET to that of Napster, there is not much difference between their underlying architectures. The main distinctions are that Napster and Gnutella are both built on top of the current Internet and offer more efficient designs and protocols.

Napster [7] was designed to be fairly centralized; all of the nodes participated in the network through a central server. When nodes would join, as part of the

process, they would submit a listing of their available music files to this central server. It would then be the only contact necessary in order to search for a specific song. The results, along with the details of the node hosting the selected song, were returned to the querier so that a direct transfer could be eventually established, by which to transfer the song file. Such an architecture allowed Napster later to place restrictions on the searchable material as well as block certain people from joining.

On the other hand, Gnutella [14] did not have any of these control features available. Since it was implemented as a totally decentralized network, there was no central point available at which to implement such filtering. This was, however, their main motivation — to put the control into the users' hands. Additionally, their design made the network more reliable. The loss of a single node, for example the Napster server, would not cause any availability issues.

Gnutella, like Napster, supported search features. Due to the lack of a central server, all searches were flooded through the network. When a node would receive a search, it would run the query against its list of available files and send back any results, while flooding more of the network with the search query. The joining process was also slightly different. A joining node had to contact a bootstrapping node within the network so that the routing tables could be updated. These features made Gnutella an uncontrollable threat. From the view of businesses, there was no easy way to control what was made and not made available on the network. Thus, further file sharing lawsuits would be deemed necessary [15].

2.3.3 Pastry

Academic institutions adopted some of Gnutella's achievements and improved on others. The first to go was the inefficient searching techniques. Flooding is a very expensive network operation as it congests the network with unnecessary traffic.

Their other primary focus was directed towards improving the overall routing efficiency of the network.

Pastry [2] was one such attempt at doing exactly this. The Pastry developers decided on the use of a ring-like network architecture, where each node exactly two neighbors (as depicted in Figure 2.3). When each of these nodes joins the network, they are systematically assigned node IDs and provided with routing information.

Similar to FidoNet, in order to help reduce the roundtrip time, it was decided to add these nodes based upon a metric of locality. If nodes were assigned IDs based upon their geographic location, two neighboring nodes in the network ring would also be geographical neighbors. The idea was that the geographically closer two nodes were, the fewer hops a packet would have to make. To reinforce this idea, the node IDs were assigned based upon a hash of the node's IP address, such that two similar IP addresses hashed to similar IDs.

In addition to an ID, each Pasty node is required to maintain some other information. First, it has to keep track of all of its neighboring nodes in what is called a `LeafSet`. Second, the upkeep of a routing table is necessary. This table is only contains part of the routing information for the entire network. All of this information, including the ID, is determined and sent to the node at the start of the Join process (Fig. 2.2). Notice that this process requires the network to properly route the join request through the network, but how do we route when the final destination is unknown?

Once the nodes are in place around the ring topology, how do they communicate? Communication occurs by way of a routing protocol. Due to the ordering of the nodes and the configuration of the routing table, all messages can reach their destination in, at most, $\log(n)$ hops[2], where n is the number of nodes in the Pastry network. An example of this routing can be seen in Figure 2.3.

[2]This metric will be experimentally validated against our Phyllo implementation in Chapter 6.

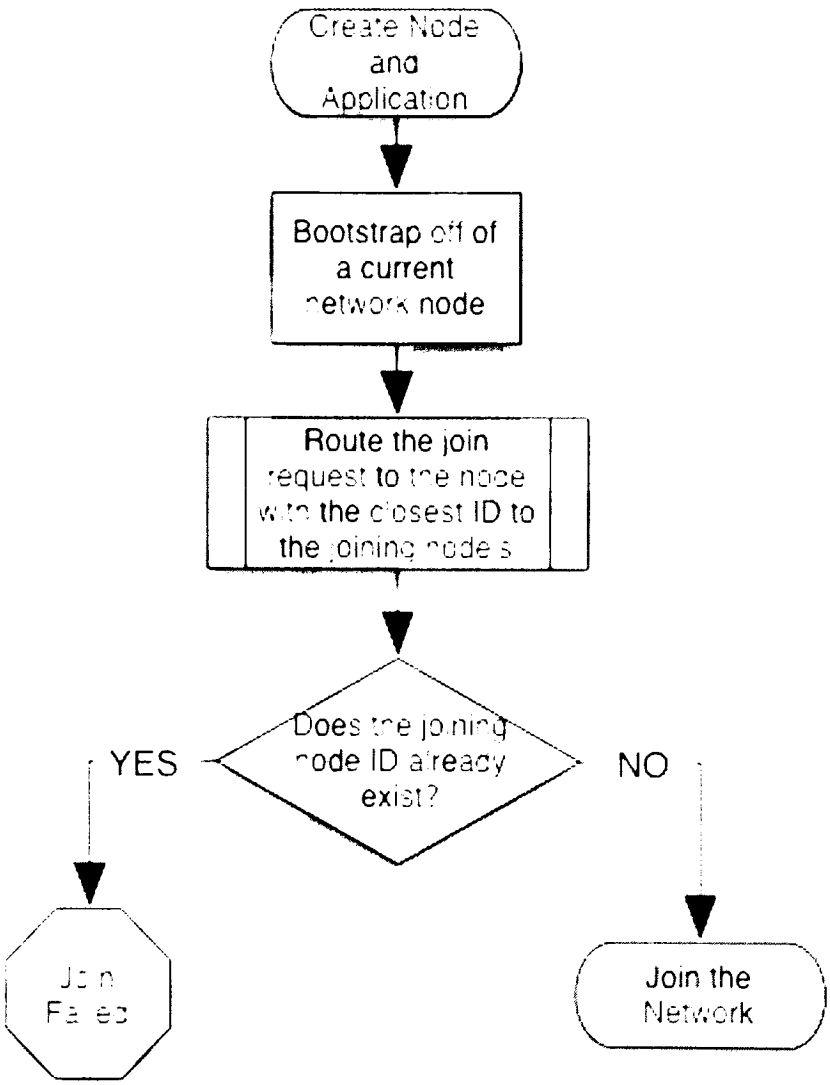

Figure 2.2: Flow chart of the Pastry Join Protocol

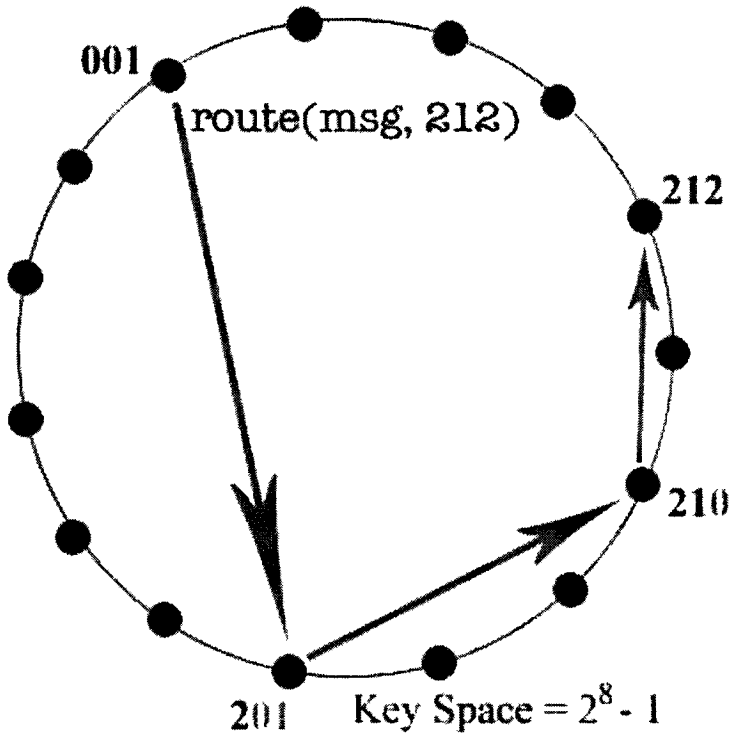

Figure 2.3: Simple Pastry Routing Example

All messages are targeted towards a specific destination ID within the boundaries of the ID space. Whenever a message is to be sent by a node (either from the message originating or being forwarded here), the node will first check the destination against the ID range encompassed by its neighboring nodes. If the destination is in this range, the message is forwarded to its final destination, the neighboring node with the ID closest to that of the destination. If it is not within that range, the current node will access its routing table. This node will then locate the node in its routing table with the closest ID to the destination and forward the message to that node. The process continues until the destination is reached or the packet is dropped because of network congestion, timeout, or similar.

To make this routing as efficient as possible, Pastry uses dynamic routing tables, referred to as `RouteSet` or `RouteTable`. This requires that nodes actively monitor

the network. They then send out routing updates to inform the rest of the network of any changes or improvements that should be made to the current routing layout. When one of these routing update messages is received, the node revises its routing information (if necessary) according to the information enclosed within the update. A similar process to this is also used when a node joins or leaves the network. An overview of this routing protocol can be seen in Figure 2.4.

This is just a brief overview as to the design and functionality of the Pastry network. Only the necessary elements of Pastry were described to allow for an appreciation of the architecture and provide enough background knowledge from which to describe their inherent security flaws. For more detailed information, the reader is referred to [2].

2.3.4 Chord

Chord [13] is very similar in design to Pastry. It uses a decentralized, ring-like topology and an architecture that improves performance and scalability metrics. Therefore, instead of reiterating a majority of Pastry's architecture, we will provide an explanation of the major differences seen in Chord. Additionally, the motivation behind such design choices will be explained.

Unlike Pastry, Chord uses a more static routing structure. By doing so, much of the overhead latency experienced with the Join process of Pastry is eliminated. Also, this reduces the amount of routing updates that are sent, thereby reducing congestion. Chord's designers argue that the simplicity of their architecture more than compensates for the minimal gains accrued from Pastry's dynamic routing.

Another difference between these two architectures is that Chord does not assume anything about the underlying network. While Pastry tries to organized nodes by

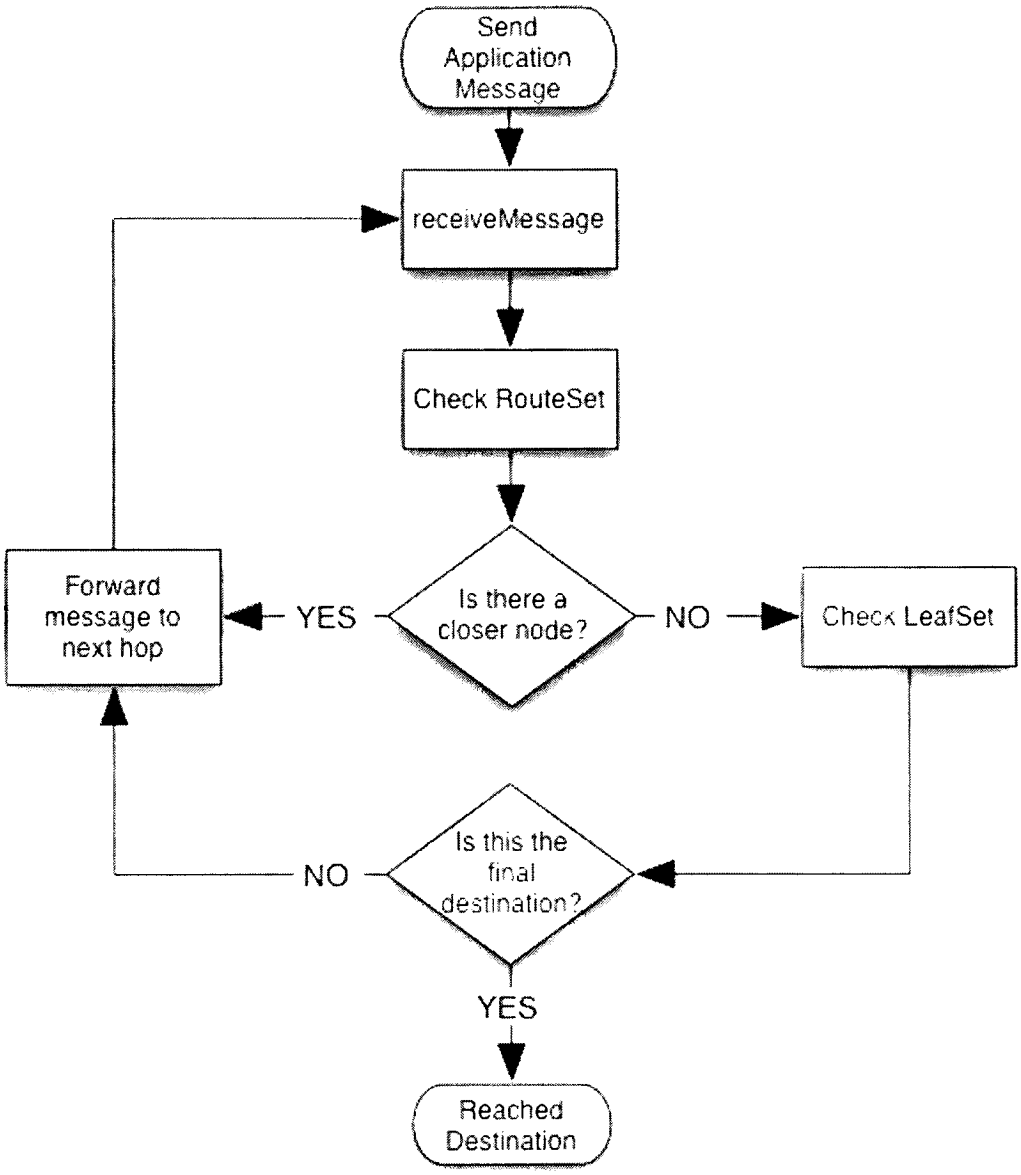

Figure 2.4: Flow chart of the Pastry Overlay Routing Protocol

their geographical distances, Chord randomly assigns the joining nodes throughout the entire ID space.

There exist many more variations of P2P applications and overlay architectures than are described here. We only attempt to highlight the necessary architectures with which to explain some of the history behind the development of the modern overlay network and its underlying security problems. Many of the other overlays, some of which have been previously mentioned, have very similar architectures to the ones described.

2.4 Related Work

There has been several previous projects that have examined the security problems surrounding P2P networks. Several of these solutions [16, 17], we viewed as unfit as they went against the openness and public nature that caused the network's success. The addition of central certificate authorities (trusted servers) to control admission or to handle the assignment of node IDs creates a single point of failure, a solitary target for attackers to focus upon. Furthermore, some of the mitigation techniques, especially those in [16], required additional protocols and massive communication overheads. Phyllo was developed to be overlay independent and easily implementable, while only suggesting that only relatively minor changes be made. Most importantly, Phyllo is transparent to the P2P application. Aside from the security aspect of P2P networks, there has been research done into various hierarchical P2P systems and peer evaluation techniques — both of which are integral parts of Phyllo.

Hierarchical architectures have already been imposed upon overlay systems [18–20]. However, the research has always been interested in its routing capabilities,

never its use in security. More straightforward implementations, such as [18], have merged multiple, distinct overlays into a unified overlay network. Select nodes from each of the original networks were chosen to act as routing gateways. While similar to our system in basic design, it is still vulnerable to the same set of overlay security problems.

HIERAS [19] attempts to decrease routing times by partitioning nodes based upon their link latency. While this does reduce transmission times, it has no impact on the network's security. Malicious nodes can still cause massive damage. Furthermore, [18] and [19] are static hierarchies; once a node has joined, the node can not move to a different hierarchical partition.

Our Phyllo framework also shares some characteristics with Jelly [20]. Jelly is a framework that arranges nodes into hierarchies based upon network loads and locality; this aims to reduce transmission times. Additionally, Jelly balances its hierarchy layers by dynamically moving nodes between the layers, similar to the promotion and demotion capabilities of Phyllo. Juxtaposed, it is evident that Jelly offers no security advantages.

Phyllo provides a framework for network pluggable node evaluation schemes (Section 5.4). Many research attempts into this area can be implemented in support of Phyllo, such as [21, 22]. Other schemes [23, 24] are not candidates for our framework, as they a require joining node to "pay its due" before fully participating in the network. We believe that this approach is in opposition to the openness tenet of overlay networks. Instead, Phyllo offers better routing performance and more responsibility for well behaved nodes. There is no punishment for new or unknown participants.

In terms of P2P security, Secure Overlay Services (SOS) [25] offers a way to mitigate denial of service attacks. It is able to provide transport-level encryption

in support of secure communications, along with access control policies and logging capabilities, for grid networks through its Peer-to-Peer Security Layer Framework (P2PSLF). P2PSLF secures a message's integrity and confidentiality, but ignores the fact that malicious nodes may just drop the protected message.

Chapter 3

Overlay Security Issues

> Never underestimate the time, expense, and effort an opponent will expend to break a code.
>
> — Robert Morris

Due to their architecture, overlay networks inherit some security issues. Many of these security problems have been documented as, and still remain, open problems. One of the main issues regarding all of these problems is that most of the proposed solutions disrupt the service features of overlay networks.

All security researchers are faced with the same balancing act — how to introduce security measures without hampering the productivity of the entire system. For most systems it involves some compromise, virus protection in exchange for higher resource usage, or user authentication for decrease production due to login times, lost passwords, etc. However, this problem is amplified in overlay architectures. As presented previously, in Section 2.3, the architectures all exploit the assignment of nodeIds to gain significant routing advantages.

3.1 ID Assignment Problem

One of the revolutionary new features of overlay networks is the distributing routing, which allows for scalable, efficient networks. This is possible through the use of the clever association between nodeIds and search keys. Additionally, due to the network layout, each node knows the best path towards any other node (or key). These are both designed to be efficient, assuming the random distribution of the IDs and keys. In Pastry, for example, the nodeIds suppose to be assigned based on the physical location of each node. Therefore, neighboring nodes in the network are also relatively close physically close.

However, this strength could also be turned into a major security hazard. If a malicious user could can control a large portion of nodes in the same area of the network, they could disconnect the routing circle of Pastry. Any messages sent through this portion of the network could be dropped, forcing each message to be resent in another direction. The more groups of nodes that an attacker controls, the more partitioned the network may become.

This issue is even further compounded when examined relative to the Sybil attack [26]. With this attack, malicious users flood a peer-to-peer network with new nodeIds such that each node is represented by several different IDs. Douceur argues that this attack can only be prevented through the use of some sort of central authority (e.g. VeriSign or ICANN) must be used. He further reasons that it is not realistic for collaborative communities, such as peer groups, to be able to identify masquerading nodes.

The use of node IDs can be seen in the newer networks — Pastry and Chord, among others — as well as in the architecture of USENET and FidoNet. Thus, a majority of them can be targeted by these attacks. However, it is easier to perform

such an attack on networks that do not randomly assign IDs. For an attacker to gain control of a large portion of a Pastry network, they would just have to join with a bunch of nodes with similar IPs (or execute a Sybil attack). Contrarily, some of the other P2P networks, such as Napster, and Gnutella, are not as susceptible. They were built to provide anonymous access, therefore many nodes are not even given keys.

To mitigate the possibility of this occurring, the network must ensure that all incoming nodes are randomly dispersed throughout the ID space. One proposed solution is to use a certificate authority to assign the IDs. This brings up a couple of problems: (1) the certificate authority is against the basic principles of decentralized networks, it provides a single point-of-failure, and it does not scale; (2) randomly assigning the nodes eliminates the potential to reduce the round-trip time (RTT) between adjacent nodes. This second issue could cause messages to be routed across the globe several times before reaching its destination. In this case, the negative effect outweighs the slight security improvement.

3.2 Message Forwarding

Problems with message forwarding have been around for centuries, dating all the way back to the Byzantine Empire. In the classical problem known as the Byzantine Generals Problem [27], the generals of the Byzantine army had to come to a unanimous decision whether or not they should attack an enemy. Since the generals are located throughout the empire, they can only communicate through messengers. How can a general be sure that the correct message was delivered?

To be sure that the messenger acted in good faith, we need to solve two problems. "Was the message delivered?" And if it was, "was it the correct message?" Only

an affirmative response to both can guarantee to the general that the messenger did not betray him. However, if the message was not received or an incorrect message was received, it does not necessarily implicate the messenger. What if the messenger died or was taken captive along the way? Or maybe the other general was not there? What if the scribe made a mistake in writing the message? If a message is lost, it may not be the messenger who is to blame.

This same situation, which was present in the days of FidoNet, is still applicable to every overlay routing architecture. It is possible for a malicious node to act as a bad messenger and deliver a bad message or no message at all. The obvious solution would be to verify through another source that the correct message did arrive. This solution, however, cause the same problem — what if the verifier lies? It is also within the realm of possibilities for the underlying network to drop the message (read: packet) because of network congestion, or for the data became corrupted.

There have been many attempts to solve the Byzantine Generals Problem, namely the Byzantine Agreement protocol [28]. This protocol allows for two generals to verify the message from a third general. However, the implementations of this protocol depend on point-to-point communications or central servers.

As an overlay solution to this problem, [16] proposes that a failure test be performed to detect whether or not the message arrived. If the test shows that the routing failed, or if the test times out, the original message can be sent using redundant routing. This method of routing will send out multiple copies of the message through various channels with the hope that at least one will arrive at the correct destination.

3.3 Routing Table Maintenance

To maximize the efficiency of the network, it is often required that the routing table be dynamic. If any connection were to die or any nodes were to join or leave the network, the routing tables must be adjusted accordingly. These dynamic tables also allow for nodes to determine the best routes, similar to many of the routing algorithms in use in today's Internet.

In order to control the table maintenance among the nodes, underlying routing update messages must be exchanged. As with the prior problems, the routing protocols assume that all of these updates are not malicious in intent. Attackers can easily take advantage of this assumption by distributing bad routing information. Without being able to verify the correctness of this data or the intent of the originating node, the receiving nodes are force to accept it, hopefully to improve the efficiency of the network.

As detailed in [16] and [29], overlays that impose weak constraints on which nodes can be in the routing table, such as Pastry, are more susceptible to these attacks. Chord, on the other hand, uses a constrained routing table. It requires that the nodes in the routing table have the closest nodeIds to a specified ID. Pastry's implementation is leveraged towards performance, while Chord focuses on security and stability. Naturally, to improve the security on all networks we should follow in the steps of Chord and use more restrictive routing techniques. Older versions of overlay networks did used very crude flooding algorithms to route information through the network. Their lack of a dependence on routing tables makes them immune to bad routing data.

3.4 Lookup Attacks

It is also possible for malicious nodes to exploit the lookup service employed in some P2P systems. This service is intended to be used by nodes to locate data. Akin to the problem of message forwarding, malicious nodes can exploit the lookup protocol.

Since in networks such as Chord, nodes can track the progress of their lookup calls, it is not enough for a malicious node to just drop the request. If this were to consistently happen, nodes may begin to suspect the malicious intent of such a node. Instead the malicious node must participate in the lookup protocol to the extent that it does forward out the lookup messages. The key to attacking this protocol is to forward these messages away from the destination or to nonexistent nodes.

From the point of view of the node that originated the message, the malicious node is viewed as a normal, participating member. The originating node has no way of knowing that the other node did not forward the packet correctly. If the lookup fails, it is attributed to packet loss on the underlying network structure. If it succeeds by way of the malicious node, the requesting node will have bad information.

So far, the only attempts being made to counter this problem are similar to those of the message forwarding attack. Therefore, the solutions involve too much bandwidth and processing overhead in order to detect.

3.5 Denial of Service

In addition to the typical denial of services attacks that can be performed on the Internet, every type of overlay network is also vulnerable. In this category we place all attacks that are concerned with reducing the network's availability. The common targets of such attacks are usually central management or authentication servers, which control the entire network. Even though these servers are not present, there

do exist several different types of attacks that can be carried out by malicious nodes that can hamper these decentralized networks.

The first type of attack is the typical denial of service attack that occurs as an effect of congesting the network with too much data. A node can simply flood the network with multiple lookup, routing, or request messages. If these messages are received faster than they can be processed, the nodes will become overburdened and other nodes cannot perform any meaningful operations. This particular attack can either target a specific group of nodes or the entire network.

Another example of a P2P denial of service attack is the continued and rapid joining and leaving of a node. Every time a node joins, the network must update itself through the bootstrapping process. With this process, the joining node is provided with the necessary routing and network information, while the other nodes are alerted to the presence of the new node. A reverse process may happen if a node leaves by informing the network of its departure. If this does not happen, the remaining nodes will find out because of dropped messages and will use routing updates to inform the other nodes. Repeatedly joining and leaving causes this whole process to exhaust network resources and reduce availability.

There exist other forms of attacks, such as the aforementioned Sybil attack, that have a side effect of harming the performance and availability of the network. As with all prior denial of service attacks, a good solution does not exist. Current research shows some promise in detecting these attacks by analyzing the network traffic. If this traffic abnormally peaks, it might be the sign of a current denial of service attack. Unfortunately, there is no additional advantage that peer networks have to try to mitigate this type of abuse.

Chapter 4

Phyllo: An Overlay Security Framework

> You cannot escape the responsibility of tomorrow by evading it today.
> — Abraham Lincoln

4.1 The Motivation

The primary drive behind Phyllo's development is not only the architectural security issues within overlay design detailed in Chapter 3, but also the effects of the exploitation of one of these flaws.

By taking advantage of the P2P routing, for example, an attackers can control the entire network. The weakness in the routing scheme allows them to filter and manipulate all of the network traffic! If present only in relatively small numbers, attackers may only be able to exert control over specific points. As their numbers increase (as compared to the size of the network), they will be able to dominate more and more of the network. If this is the situation if the attackers operate independently of one another, just imagine the potential if they worked together...

4.2 The Framework

The Phyllo overlay security framework is designed to adjust the underlying architecture of the various overlay networks. Its goal is to be small and easily adaptive to all overlay network architectures, while providing increased security with minimal communication overhead. Phyllo can be divided into three separate parts: (1) the structure/topology changes, (2) routing modifications, and (3) the promotion and demotion protocols. The topology and routing changes are relatively straightforward and are presented in Sections 4.2.1 and 4.2.2. Part #3, the promotion and demotion protocol, is a new concept unique to Phyllo. It allows for the controlled movement of nodes between the various sections of the topology and is introduced in Section 4.2.3.

4.2.1 Phyllo Structure

To accomplish such a seemingly daunting task, Phyllo uses an adapted hierarchical structure. This structure can be easily adapted for most topologies; the only requirement is that the nodes must be identified by unique tag, which must have a natural ordering. For example, Phyllo can be used on the ring-like structure of Pasty [2] or Chord [13]. A visualization of such a topology can be seen in Figure 4.1.

The name "phyllo" describes the leaf-like image created by the introduction of the hierarchical topology. When viewed on a Pastry-like ring topology, the minor partitions give the appearance of leaves branching off of a tree trunk (Fig. 4.1). The term "phyllo" is a prefix, derived from Greek, meaning "leaf". Furthermore, the name pays homage to the Pastry [2] overlay network on which Phyllo was initially implemented.

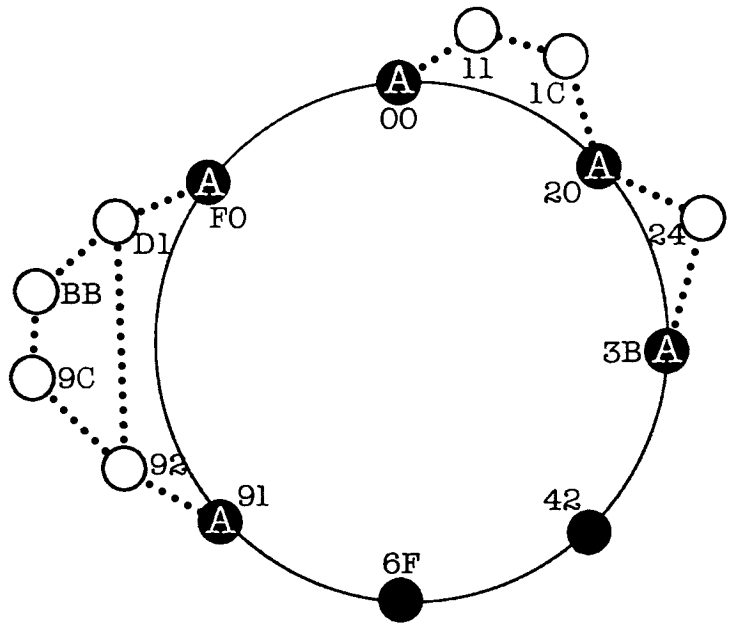

Figure 4.1: Phyllo hierarchy on a P2P ring topology. The minor and major nodes are colored in white and black, respectively. The A denotes a major, anchor node.

Definitions

The security advantages of Phyllo are derived from this hierarchical topology. However, before describing these advantages, it is first necessary to formally introduce some terminology that will be used throughout the remainder of this paper. First off, there is the notion of a *node*, or a single member of a network; there is not necessarily a one-to-one correspondence between nodes and computers, as a computer can be simultaneously acting as several members of a network. When the nodes join to the P2P network, they are added to one of the hierarchical divisions within Phyllo. A division is referred to as a *partition*. The number of partitions and how to decide to which partitions the nodes are added is a problem that is left up to the implementor of the network.

In this case, since we are dealing with security, we divide the Phyllo network into two partitions and a node's partition membership is based upon how much we trust it — will it act for the good of the entire network, or will it try to be selfish? We refer to these two partitions as the *major partition*(e.g., nodes 00, 42, and 91 from Fig. 4.1) and the *minor partition* (nodes 00, 20, 3B, 91, and F0). It is important to note that all Phyllo partitions use the same ID space, and the membership within these partitions is exclusive. A node must belong to exactly one partition. If the node is part of the major partition, then we call that node a *major node* (e.g., nodes 00, 42, and 91 from Fig. 4.1). Likewise, if it is in a minor partition, the node is a *minor node*: nodes 1C, 24, 9C, etc.

In addition to these two node types, we also define a third node type, an *anchor node*. Anchor nodes are members of the major partition, but have the additional responsibility of being linked to at least one minor partition. They act as the link between the partitions, determining what information can be exchanged.

Hierarchical Structure

The security benefits gained from the hierarchical layout, stem directly from these anchor nodes. When the anchor nodes server as the middleman, they also function as a filter or firewall. This effectively divides the partitions into to independent networks that happen to share the anchor node(s); nodes in one partition are never made aware of the existence of any other nodes outside of that node's partition. Additionally, they can be used to monitor the activities of the nodes of the minor partitions to which it is linked. This information can be later used as input in order to make generalizations about the intent of a node, based upon the anchor's interaction with that node. These generalizations can be later used by the promotion or demotion protocol to move a target node into or out of the major partition. The

monitor also acts as insurance that an anchor does not abuse its power, since it is being monitored by neighboring major nodes.

4.2.2 Phyllo Routing

At the partition level, Phyllo does not make any changes to the routing protocol of the original network (e.g., Chord, Pastry, CAN). It is only when the routing is viewed from the network's perspective that Phyllo's presence is made evident.

Phyllo distinguishes between two different P2P message types. An *application message* comes from the P2P application utilizing Phyllo. These messages allow the users of the application to perform various tasks such as searching, responding, and transferring files (in the case of a filesharing application). The other type of message is that which is necessary in order to perform typical network maintenance. These *overlay messages* are static and defined by Phyllo or the underlying overlay network implementation, whereas application messages are define within the P2P application.

Overlay Message Routing

Since overlay messages are necessary for the network's operation, their control is vital to the maintenance of the hierarchical structure. Though important, this is only of secondary concern within Phyllo, for it is this message type that allows for an attacker to execute the Routing Table Maintenance attack (Section 3.3).

Generally, the overlay messages are not routed; they are sent directly to a node's neighbors. Since the information contained within is only applicable locally, there is no need to "flood" the updates to the whole network. Phyllo maintains the same approach when dealing with overlay messages. However, Phyllo provides a more stringent definition to being a "node's neighbor."

Expanding upon the standard notion of neighbors (i.e., "the n [numerically] closest nodes to a given node"), Phyllo adds the additional restriction that the nodes must all be members of the same partition. This is where the anchor nodes, and the fact that they are members of both the major and adjoining minor partitions, comes into play. It is the responsibility of the anchor nodes to act as border guards, ensuring that the overlay messages do not cross the partition boundaries. Instead of blocking the communications as they occur (similar to a firewall), the anchor nodes just verify that the routing update overlay messages do not specify nodes outside of the recipient's partition.

Application Message Routing

All of the changes to the underlying application message routing protocol (Figure 2.3) are done in support of the hierarchical structure of Phyllo. The addition of the security partitions make the routing a little more complex. While transmitting a message through peers, towards the one closest to the message's destination, the message may have to cross the partition boundaries up to two times. Therefore, the anchor nodes must be able to tell when to continue to route in the major partition and when it is necessary to send it to one of the adjoined minor partitions.

Simply, this is done through a comparison between the current anchor node's ID and its neighboring anchor node's ID (assuming a minor partition is present between them). If the ID is between them, the message will be sent to that minor partition. In any other case, the message remains in the major partition. Such a check is necessary because a minor partition may grow to a size where every member cannot be tracked by the anchor nodes.

To further explain this routing, we will refer to Figure 4.2, which shows a message being routed between two different minor partitions. To simplify the understanding,

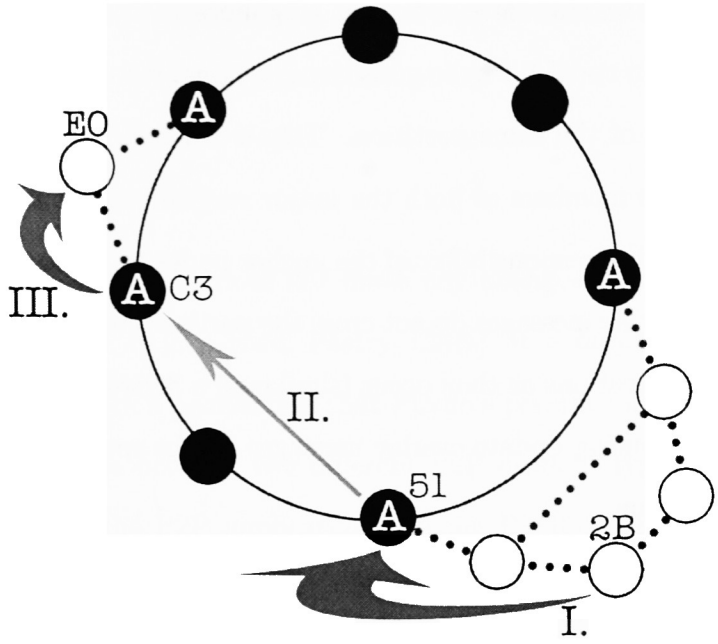

Figure 4.2: The Three (3) Stages of Minor-to-Minor Node Phyllo Routing

the routing process is broken down into three stages. Initially, a message sent from a minor node is routed through that node's partition towards the destination address. Eventually, it reaches the anchor node closest to that destination (Stage I). The next stage (Stage II), involve the message's continued routing through the major partition. At some point, it arrives at the major node with the closest ID to the message's destination. Here, if this major node is not an anchor node, the routing process is complete and message is at its target. Otherwise, the major node is an anchor, and the message's destination is checked against the range of its minor partition IDs. If the destination is in that range, the message is transitioned into, and routed within, that minor partition (Stage III).

A flow chart diagraming the entire application message inter-partition routing can be found in Figure 4.3.

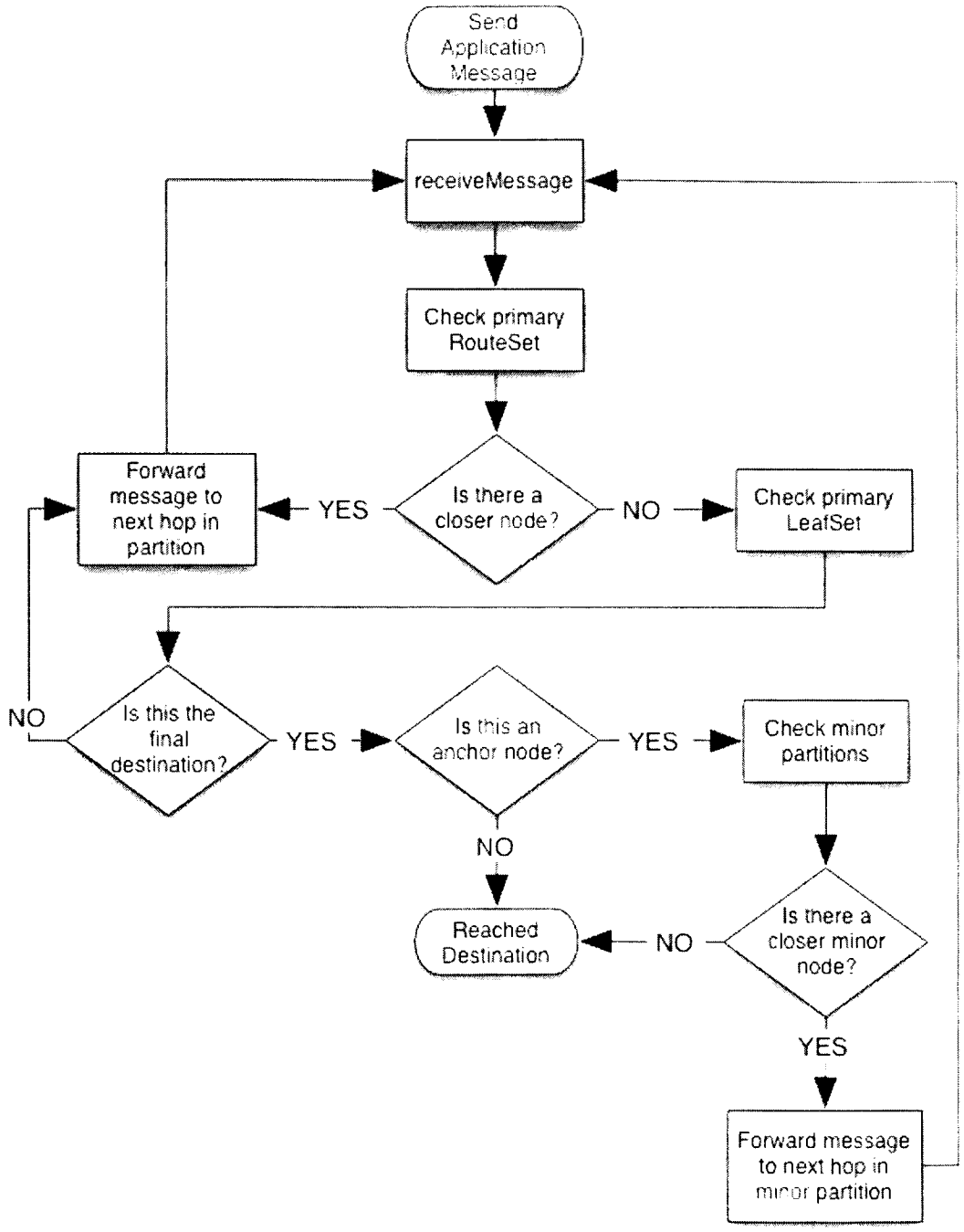

Figure 4.3: Flow chart of the Application Message Inter-partition Routing Process

4.2.3 Phyllo Promotion and Demotion Protocols

The revolutionary feature of Phyllo, is it's ability to dynamically move nodes between hierarchical partitions. This allows for the control of nodes based on their behavior, or other scheme, which can be declared by the implementor of the overlay network. Instead of a more static evaluation, Phyllo's promotion and demotion protocols recognize that the variable of time is important in the realm of security. A computer system maintenance is a cyclic process — a machine may be controlled by a remote attacker or virus at one moment, then reverted to a normal, localized control by an attentive system administrator or actions taken by anti-virus software or intrusion detection systems (IDS).

It is important to understand that there must be some underlying configuration and implementation in order to tailor the promotion and demotion protocols to suite the requirements of the P2P application. Every network application is designed to target very specific use cases, and thus any supporting systems must also be adapted to fit these goals. For example, a backup banking network, which might operate over an ad hoc conglomeration of various banking institutions, is going to require very stringent controls, preferring to be on the secure side of the security vs. usability scale. Meanwhile, a P2P application to support open data sharing between organizations and researchers worldwide, may be interested in preserving some minimal control over the nodes to ensure some level of data integrity.

Phyllo has a trust evaluation framework in place that allows for such a granularity of control. The accuracy of evaluation of an implementation of this framework is describe in terms of α. α is a percentage indicating how often the evaluated metric is correct. An α value of 90% means that the associated trust evaluation system detects malicious (non-trustful) nodes correctly every 9 out of 10 tries (or has a failure rate of 1 in 10). The tradeoff here is that to achieve a better accuracy,

more information will have to be exchanged and stored, which reduces the network scalability and bandwidth.

The goal of the framework is to be flexible enough to support any desired means of evaluation, including the P2P trust assignment schemes presented in [21, 22, 30]. Implementation details, as well as potential uses, of Phyllo's trust evaluation framework will be further discussed in Chapter 5.

Promotion Protocol

The Promotion protocol servers to improve the efficiency of the network. As will be demonstrated in Chapter 6, the average hop count is minimal when all of the nodes are contained in the major partition. Therefore, Phyllo would like to promote every node as soon as its trust evaluation metric is sufficient. Additionally, Phyllo's architectural can lead to network bottlenecks occurring at the anchor nodes. The promotion of a node leads to one more anchor nodes as well as the splitting of a minor partition. Not only can promotion help with security, but also with improving network efficiency.

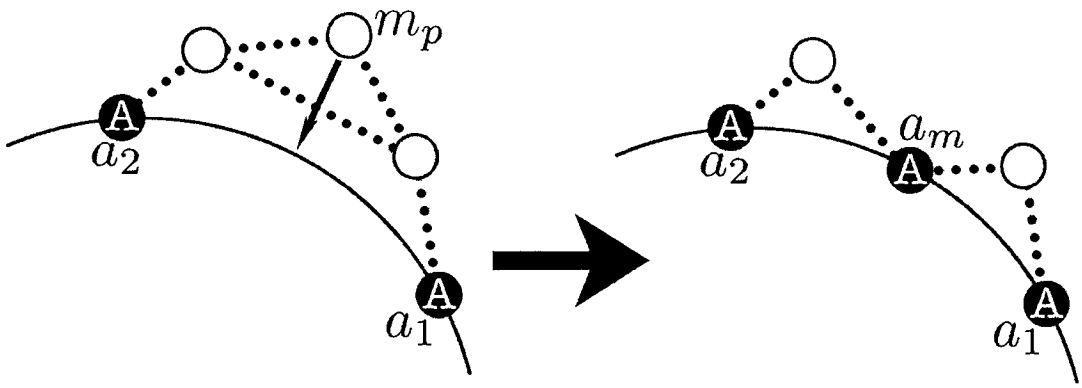

Figure 4.4: Phyllo Promotion Protocol

The promotion protocol operates as follows (referencing Figure 4.4):

1. An anchor node a_1 scans the trust evaluation ratings of the minor nodes connected to it.

2. a_1 selects a minor node m_p for promotion, based on the minor node's high rating.

3. a_1 informs m_p and the opposing anchor node a_2 of the impending promotion.

4. m_p receives the promotion message and splits its list of neighbor nodes into its two new minor partitions: nodes with IDs between a_1 and m_p go into one minor partition, and the ones with IDs from m_p to a_2 go into the other partition.

5. a_2 receives the promotion message and updates its minor partition that contains m_p — only the nodes with IDs between m_p and a_2 are kept, the others are deleted. a_2 sends the updated partition and routing information to that minor partition.

6. a_1 likewise updates its minor partition, keeping only the nodes between itself and m_p. a_1 sends the updated partition and routing information to that minor partition.

7. a_2 circulates the updated major partition information, with m_p's promotion, to the rest of the major partition.

After this process[1] is complete, m_p has been promoted to an anchor node a_m and the old minor partition between a_1 and a_2 is now split.

Demotion Protocol

Phyllo's Demotion protocol acts in opposition to the Promotion protocol; everything done with promotion, can be undone with demotion. The main reason for including a demotion protocol is to handle any nodes that exploit the Promotion protocol by behaving in accordance to the network's goals until they are promoted. Once the promotion would occur, Phyllo could no longer offer any protection. The Demotion protocol is Phyllo's last line of defense against intelligent attackers or the compromising of systems hosting major nodes. Furthermore, a loss of any anchor

[1]Promotion protocol steps #4-7 occur in parallel, so their actual sequence in practice will vary.

node can be viewed as a demotion of that node, alleviating the problem caused by such a dynamic network.

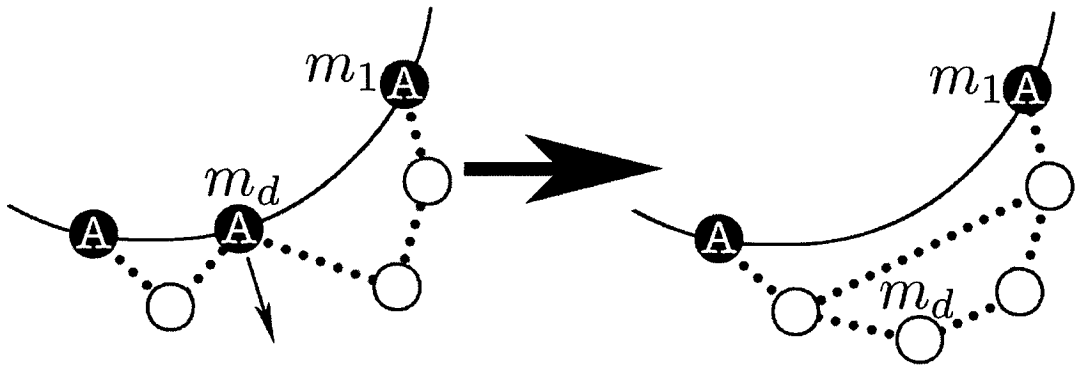

Figure 4.5: Phyllo Demotion Protocol

The Demotion protocol operates in the same manner as the Promotion protocol, but performs the converse of each operation. The Demotion protocol (Figure 4.5) operates by:

1. A potentially harmful major node, m_d, is identified by another major node m_1.
2. m_1 notifies the neighboring major nodes, including m_d, of m_d's upcoming demotion.
3. When m_d receives the message of its demotion, it removes the neighbor nodes from its current (major) routing table and moves each of its minor partition's routing information to the primary routing table. m_d then notifies this new partition (which includes m_1 and m_2) of the new routing changes.
4. The other major nodes, when they hear of node m_d's demotion, simply remove m_d from their routing tables. If m_d is bad, they do not want to even know of it.

Upon completion, the common minor partitions between m_1, m_2, and m_d are joined into one minor partition, with m_1 and m_2 as anchors. m_d is now a minor node in that minor partition.

Disobedient Nodes

Both the Promotion and Demotion depend on some cooperation from the nodes. If some of these node are not trustworthy, how can we rely on them following the protocols? Simply put, we cannot. However, the way the protocols are designed, disobedient nodes can only bring harm to themselves.

This feature comes from the way Phyllo assigns the minor and major tags. A node is either major or minor depending on how another node views it. A minor node is viewed by another minor node as being a major node, since it has no knowledge of the major partition. Whenever a node receives a message, it checks the message's last sender and handles the message with respect to how it views that transmitting node.

What does this gain us in terms of disobedient nodes? This means that if a node tries to fool the protocol, its neighbors will still be informed of the promotion or demotion. If the node attempts to act differently from how its neighbors view it, all of its messages will be dropped because the sending node is not known by the recipient. This type of attempted exploit is easily detectable, but Phyllo does not current support any way of punishing or removing the misbehaving node (apart from its promotion or demotion).

Even in an absolute worst case scenario when the protocols fail, Phyllo will be effectively removed. The overlay network would lose its hierarchical structure and would function as the underlying overlay implementation (e.g., Pastry, Chord, CAN). Phyllo can never be exploited to decrease security, beyond what it was without the framework in place.

Chapter 5

Implementation: Phyllo Pastry

> This is my answer to the gap between ideas and action — I will write it out.
> — Hortense Calisher

Phyllo was implemented on Rice University's Pastry Overlay Networking Substrate 1.4 [2] for Java, version 1.4.2. The programming was done primarily with eclipse IDE 3.0.1. The final Phyllo package was released as `Phyllo.jar`, which consisted of the `rit.secure.pastry` Phyllo package and a few experimental applications.

This section will dig into the detail of the actual Phyllo implementation on Pastry. It is not necessary to understand the contents of this chapter in order to appreciate Phyllo's design and improved security capabilities. The following sections will elaborate on the internals of the Pastry-specific adaptation of the architecture and protocols discussed in Chapter 4.

All objects referenced in `typewriter` font should be assumed to be part of Phyllo's `rit.secure.pastry` package, unless otherwise specified. The *italic* font refers to basic Pastry objects found in Rice University's Pastry implementation (`rice.pastry` Java package).

5.1 Hierarchical Structure

The actual Phyllo node is located in `rit.secure.pastry.SecPastryNode`. This object contains only the core logic necessary to exist (has an ID and can store overlay data) and may, optionally, execute a thread (`rit.secure.pastry.SecPastryNode.PromotionEvaluator`) to interface with the trust evaluation implementation. This thread will periodically scan through the data gathered from the messages and other sources, and recommend nodes for promotion or demotion.

The network data that each node stores consists of a routing table and leaf set for each partition it is a member of. A leaf set stores the closest nodes, in terms of locality, while the routing table saves references to nodes elsewhere in the partition. Phyllo uses the term "primary" to refer to the leaf set or routing table for the partition to which a node belongs. Anchor nodes will also have one or two "minor" sets of information — one for each attached minor partition.

The protocols surrounding the transmission of these leaf sets and routing tables is declared within `rit.secure.pastry.standard` nest in `PartitionLeafSetProtocol` and `PartitionRouteSetProtocol`, respectively. Both use overlay messages to disseminate the necessary information only to their partition. The *RouteSet* protocol is primarily used for more efficient routing over a large number of nodes. Since Phyllo partitions the nodes, Phyllo does not benefit as much as Pastry from this data. The *LeafSet* protocol informs the network of some change: (1) a joining node (Section 5.3), (2) lost or dropped node, or (3) a node receiving a promotion or demotion (Section 5.5), are just a couple of uses.

Since Phyllo's hierarchical structure was designed specifically with the ring network topology in mind, there are no surprises with the implementation; each node just has to keep track of some additional sets of information. However, to make

the partitioned structure possible, there needed to be some changes made to the underlying routing and join protocols of Pastry.

5.2 Routing Protocol

As stated previously, there are two types of messages in Phyllo: overlay messages and application messages. There is no need to route overlay messages, since they are send directly from a node to one of its neighbors. However, all of the application-specific messages need to be routed from the application, down through the local node, then through the network (multiple intermediary nodes), and finally to its destination node and application.

The resultant Phyllo routing algorithm (Fig. 4.3), as implemented in `rit.secure.pastry.standard.PartitionRouter`, is as it was described in the previous chapter! For each received *RouteMessage*, Phyllo's router determines the best next hop based upon the destination. The method `partition.PartitionLeafSet.mostSimilar()`, which was adapted from Pastry's *LeafSet* to include the extreme nodes in the leaf set, performs this comparison.

Figure 5.1 provides a much more detailed look at the application message routing in Phyllo-Pastry. A P2P application creates a message and starts the routing process towards a chosen destination (step 1). That message gets passed through the *PastryAppl* abstract class where it is embedded in a *RouteMessage* wrapper (step 2). From here, the *RouteMessage* is sent through the local node receiving methods (steps 3–4) and ends up in the *MessageDispatch* center (step 5). Then (step 6), the `standard.PartitionRouter` finds the best next hop. At this stage, Phyllo decides whether the route message should remain in the current partition or

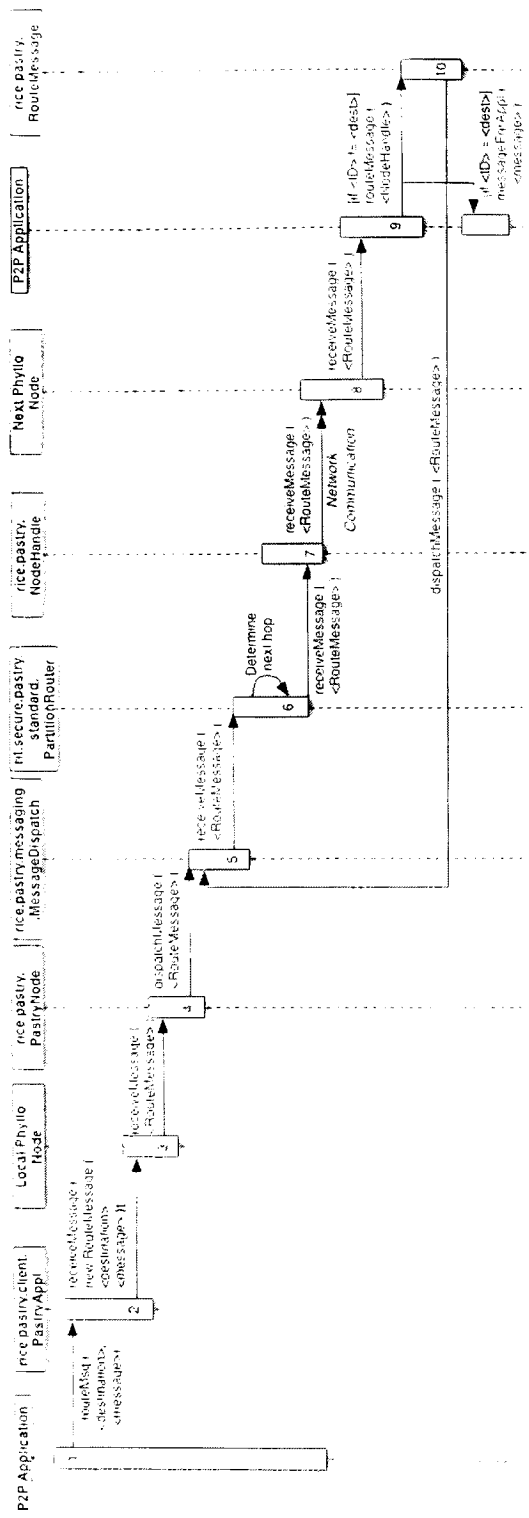

Figure 5.1: Sequence Diagram for a Application Protocol Routing

switch to another. The `PartitionRouter` then sends the updated *RouteMessage* to the local *NodeHandle* to be forwarded.

At step 7, the message is finally transmitted to the next node, who passes it to a local instance of the P2P application (step 8). At the next step, the protocol checks to see whether the received route message is destined for the current application. If it is, the message is left with the application for processing and the routing is complete. However, if it is not, the *RouteMessage* is passed back down to the local *NodeHandle* through `rice.pastry.RouteMessage.routeMessage()` (step 10), and the routing process returns to step 5 to route the message on to the next hop.

5.3 Join Protocol

Like the routing protocol, Pastry's join protocol had to also be modified to support Phyllo's partitions. Though the protocol itself is fairly simplistic, within Phyllo it has some very important responsibilities. The entire routing protocol, as described previously, depends on a set of assumptions. It is up to the join protocol to ensure that the network is built in such a manner that these expectations are met.

The process flow of the partition join protocol in Phyllo, as implemented in `rit.secure.pastry.standard.PartitionJoinProtocol` is presented in Figure 5.2. Note that the third stage, "Route the join request to the node with the closest ID to the joining node's", makes use of the application level routing detailed in the previous section. Upon initial reception of a join request, the request is encapsulated in a *RouteMessage* and routed towards its new position in the network.

At the termination of the join process, one of three outcomes will be reached: (1) allow the node to join the major partition, (2) allow the node to join a minor partition, or (3) deny the join request. There are only two sets of circumstances,

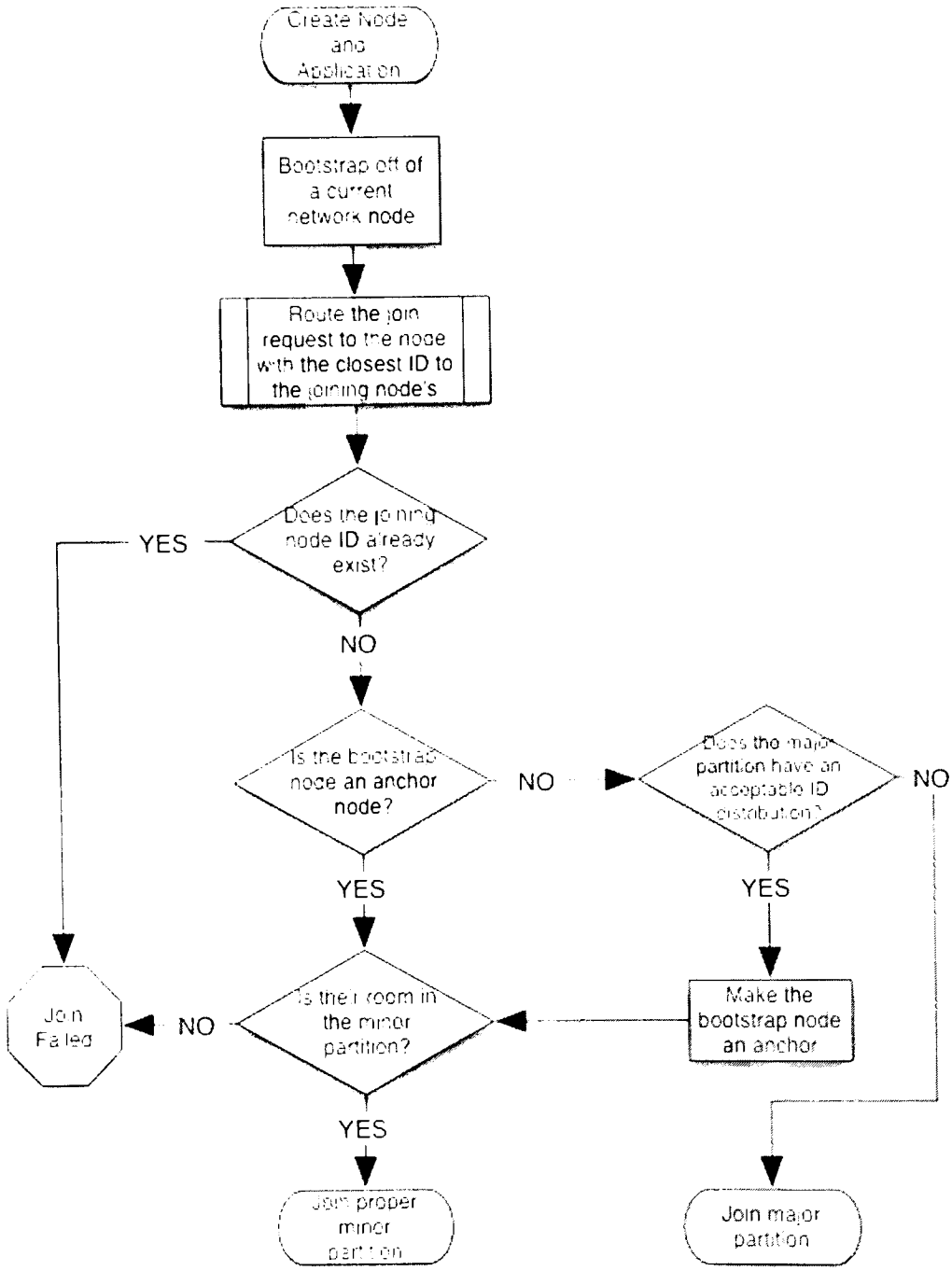

Figure 5.2: Flow chart of the Phyllo-Pastry Join Protocol

under which a node is not permitted to join the network: (1) the ID that the node is attempting to join under already exists, which is a precondition of Pastry to ensure proper routing; or (2) there is no room in the minor partition, as limited by Phyllo and the trust evaluation framework (see Sect. 5.4).

If the node is allowed to join, the partition it joins into is decided based upon some requirements stipulated by Phyllo. When dealing with a circular ID space, there is no easy way to compare IDs. There is no reliable way to perform logical comparisons of one node's location to another. For example, in an ID space of $0xFF(255)$, is $0x60(96)$ less than or greater than $0x7B(123)$? How about $0xFE(254)$? Since we are in a circular space, $0x60$ is both greater than **and** less than $0x7B$, and the same goes for $0xFE$!

If we envision the ID space as a full circle, we can talk about node relationships as being clockwise or counterclockwise from one another. This is all the further Pastry needs to go, but Phyllo needs a little more. For partitions to work, for every major node, there must be a major node located counterclockwise from it and another located clockwise. To avoid the circularity problem, a node (e.g., $0x7B$) is clockwise from a given node ($0x60$) only if the node exists on the clockwise half of the circle when divided through the given node (see Fig. 5.3), likewise for a counterclockwise node ($0xFE$). Therefore, nodes with the ID on exact opposite locations in the circle are neither clockwise nor counterclockwise from one another.

The solution is that the join protocol must setup the network per these requirements. Namely, nodes cannot be added to a minor partition too prematurely. The join protocol must check to ensure that there is proper ID distribution in that section of the network before deciding whether or not to add the new node to a minor partition. This only needs to occur at the highest partition level, the major partition. If it is assured there, then it is guaranteed at every partition beneath it.

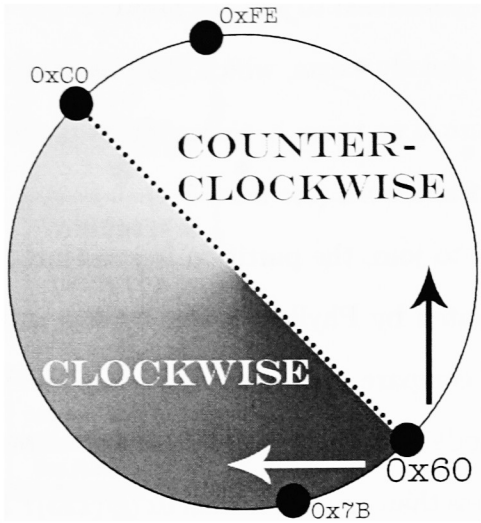

Figure 5.3: Comparing nodes in circular ID spaces

5.4 Trust Evaluation Framework

The trust evaluation framework exists to allow different overlay networks to tailor the promotion and demotion protocols to their needs. One network could use a static major partition (a la Napster [7]), another network might have a list of privileged nodes that can be promoted, while a third wants the network to configure itself dynamically to maximize routing efficiency. Generally, the framework was designed to be flexible and not require additional network overhead.

A trust evaluation framework is rated based upon its α value — the percentage of nodes that it correctly identifies (as being either malicious or trustworthy). Since this is a largely hypothetical value, Phyllo has the ability to simulate various α values. For example, `rit.secure.pastry.testing.TestRouteTableDurability` supports a simulated α value; the value can be specified on the command line using the `eval` flag:

```
bash$  java TestRouteTableDurability -eval 99
```

This will tell the trust evaluation framework to run with $\alpha = 99\%$ accuracy.

The actual framework, as implemented by Phyllo within Pastry, consists of two interfaces: `rit.secure.pastry.promotion.PromotionEvaluation` and `rit.secure.pastry.promotion.ForcedPromotionEvaluation`. The latter consists of only three methods for use only with the forced promotion protocol. These methods include: (1) `getBestNode(Id,boolean)`, (2) `splitMinorPartition()`, and (3) `promoted(NodeHandle)`. `getBestNode()` finds the node with the best trust metric rating in the specified minor partition with an ID closest to that passed in as a parameter. Ideally, this ID should be at the mid-point of the range covered by the partition. This will achieve the maximal division, assuming a purely random ID distribution. The `splitMinorPartition()` method will check to see if either minor partition, if they exist, should be split. Finally, `promoted(NodeHandle)` communicates to the framework that the node with the specified *NodeHandle* was just promoted. From there, the framework implementation might need to perform some updating of its trust statistics.

The other interface piece, `promotion.PromotionEvaluation`, takes care of the standard promotion and demotion protocols. There are nine total methods, which include some noticeable overlap from the `ForcedPromotionEvaluation` interface: (1) `allowJoin(NodeHandle)`, (2) `evaluate(Message)`, (3) `getDemotionNode()`, (4) `getPromotionNode()`, (5) `haveDemotions()`, (6) `havePromotions()`, (7) `demoted(NodeHandle)`, (8) `promoted(NodeHandle)`, and (9) `remove(NodeHandle)`. The `allowJoin` method is used by the join protocol in Phyllo to test to see if a node should be allowed to join. Methods 7–9 are used to inform the framework of any network activity. The `havePromotions()` and `getPromotionNode()` methods are used to begin the promotion protocol, testing whether there are nodes to be promoted and

then retrieving those nodes; `haveDemotions()` and `getDemotionNode()` perform the same function, but for the demotion protocol. Last of all, the `evaluate(Message)` method is called to allow the framework implementation to examine the contents of a just-received message. This is so a framework can perform some evaluations to ensure that the message has been properly forwarded, has the correct contents, etc.

These interface methods are explained further in the associated JavaDocs found in Appendix B. Examples of implemented trust evaluation schemes can be found in `rit.secure.pastry.testing.MinorPartitionEnforcement` and `rit.secure.pastry.testing.BasicEvaluation`. The first instance primarily makes use of the `ForcedPromotionEvaluation` interface to promote the most qualified node only when the minor partition reaches maximum capacity. The other model allows for node promotion and demotion based upon evaluating each received message to ensure the message was properly forwarded.

Some research has already been done into evaluating overlay nodes, and the trust evaluation framework is flexible enough to support implementations from any related research. Examples include incorporating trust management schemes of the EigenTrust algorithm [22] or the global trust model proposed in [21]. Nielson, in [30], describes a few solutions to defend against "self-interested nodes" — nodes, such as those in participating in a Byzantine Generals-type attack, that do not act for the good of the overall network. These are all valid uses of our evaluation framework. Additionally, techniques not developed specifically for P2P networks, can be adapted for use within Phyllo. One instance would be to use concepts for gauging the trust value of Usenet authors [31].

5.5 Promotion & Demotion Protocols

Phyllo can make use of this Trust Evaluation Framework for one of three tasks: (1) promoting a node, (2) demoting a node, or (3) forcibly splitting a minor partition – the forced promotion. The first and last of these tasks are taken care of by the Promotion Protocol, while the remaining one is the responsibility of the Demotion Protocol. Since Pastry makes use of leaf sets to manage the information affected by moving nodes between partitions, we built the actual promotion and demotion logic directly into the leaf set protocol, aptly named `rit.secure.pastry.standard.PartitionLeafSetProtocol`.

Each of these protocol are "pull" protocols. I.e., it is up to the Phyllo node to query the trust evaluation framework and initiate the desired process. There is no current way for the nodes to be informed of a node reaching the promotion threshold, or similar. Due to this, each Phyllo node has an evaluation thread, `rit.secure.pastry.SecPastryNode.PromotionEvaluator`, which interacts with the trust evaluation implementation at periodic intervals.

5.5.1 Promotion

A promotion is accomplished by sending a `partition.PartitionBroadcastLeafSet` message of the type `PartitionBroadcastLeafSet.Promotion`. Within this message, the anchor node that is initiating the promotion specifies the target node being promoted and will attach the new major, or primary, leaf set. This new leaf set already includes the information of the node being promoted.

The promotion protocol is best explained as a series of steps, as represented in Figure 5.4. We will make use of this sequence diagram in order to explain each

portion of the protocol. The basic procedure is as explained in Sect. 4.2.3 but is tailored to this implementation of Phyllo on Pastry.

In step 1, a Phyllo node queries an implementation of the `rit.secure.pastry.promotion.PromotionEvaluation.havePromotion()` method in the trust evaluation framework to see if there are any nodes deserving of a promotion. If the framework returns an affirmative response (step 2), the node requests the *NodeHandle* of the node most qualified for promotion by way of `PromotionEvaluation.getPromotionNode()`. Once that information is returned (step 3), the node starts the promotion protocol, `SecPastryNode.promote(NodeHandle)`, which creates a new `partition.PartitionBroadcastLeafSet` message. In this message the node states the targetted *NodeHandle*, the current major leaf set, as well as the fact that this is a promotion — `partition.PartitionBroadcastLeafSet.Promotion`. Step 4 concludes with the node transmitting the message to its *LeafSetProtocol*, `standard.PartitionLeafSetProtocol`.

This protocol, in step 5, updates the local *LeafSet* with the promoting node's information. It then informs the node being promoted and the neighboring anchor node also effected by this promotion. The protocol uses step 6 to inform its effected minor partition of the current promotion, so that they may update their leaf sets[1].

Concurrently, the promotion target and opposite anchor node must maintenance their *LeafSets*. The anchor node, in step 7, mimics step 6 in order to inform the rest of the effected minor nodes. From here, the second anchor node informs the rest of the major partition of the newly promoted node. The promotion is complete once the promotion target finishes step 9 by dividing its previous primary (minor) leaf set into two minor leaf sets — all nodes counterclockwise from it go into one leaf set, while the remainder, clockwise nodes, fall into the other minor leaf set.

[1]This is not the standard leaf update referred to previously. In a normal update, information can only be added or overwritten. In this case, it is necessary for information to be deleted.

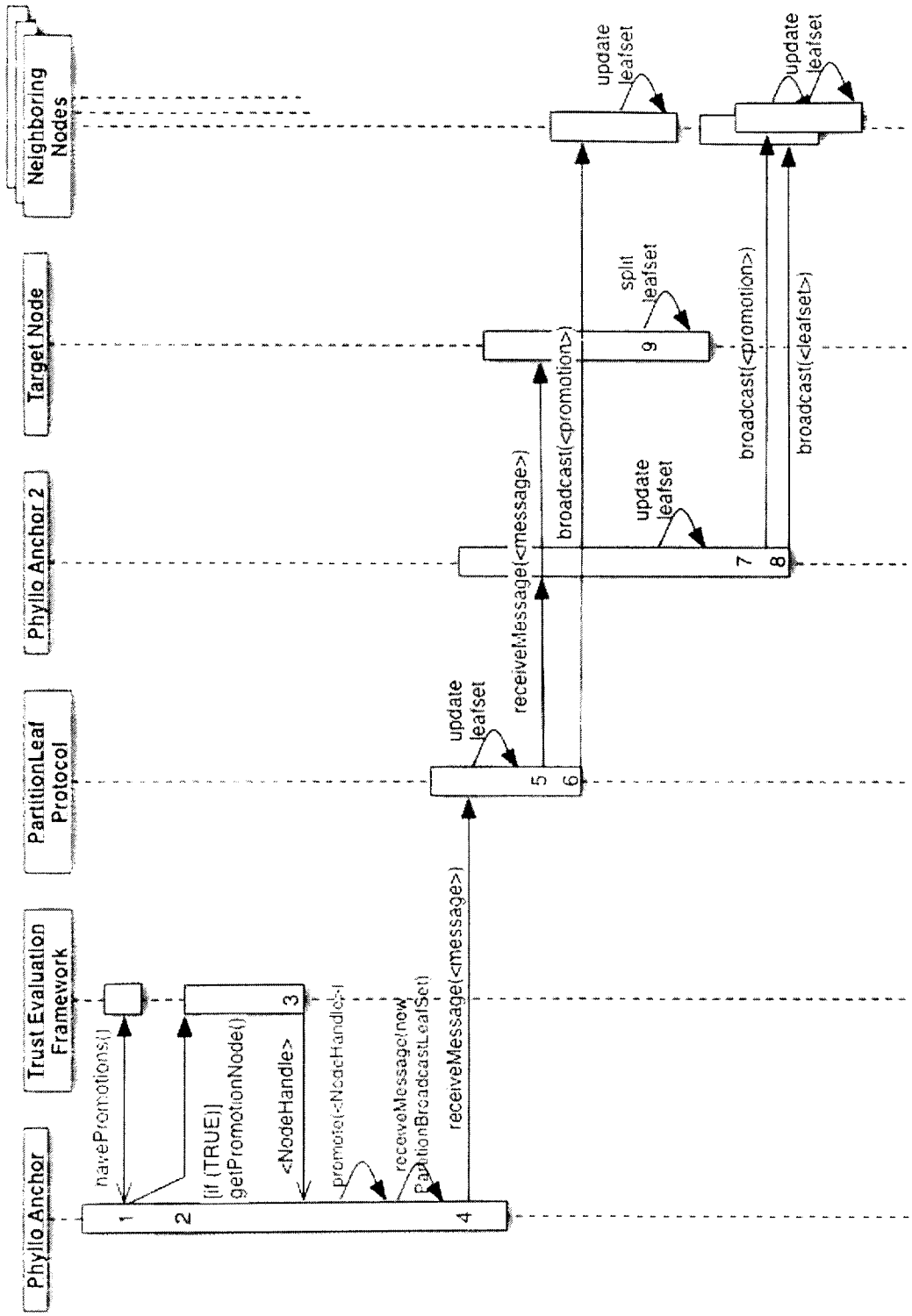

Figure 5.4: Sequence Diagram for a Promotion within Phyllo

5.5.2 Forced Promotion

Phyllo's Force Promotion Protocol is very similar to its regular promotion protocol. The only differences are that it interacts via the `rit.secure.pastry.promotion.ForcePromotionEvaluation` interface and can promote nodes that do not meet the promotion threshold requirement. However, it is only used in cases where it takes too long to wait for a node to be promoted normally. Making room in a minor partition to accommodate a joining node or splitting a minor partition to relieve a traffic bottleneck at an anchor node, are two viable instances to force promote a node.

Examining this protocol's sequence diagram (Figure 5.5, it is evident of how closely related the two promotion protocols are. To transition from the standard promotion protocol, at step 1 a different call, `ForcePromotionEvaluation.splitMinorPartition()`, is used. If the return value of that call specifies a minor partition, then the anchor node uses the `SecPastryNode.forcePromote()` method to initiate the force promotion process. Step 3, an additional call to the trust evaluation framework, from the standard promotion protocol is not necessary for the forced version. From here, steps 4–9 happen exactly as described in Section 5.5.1.

5.5.3 Demotion

Now that we have thoroughly detailed the abilities of Phyllo to promote nodes, we need to provide an explanation of the demotion side of these protocols. Like its promotion counterpart, the demotion protocol rests in `rit.secure.pastry.standard.PartitionLeafSetProtocol` and uses the `promotion.PromotionEvaluation` interface.

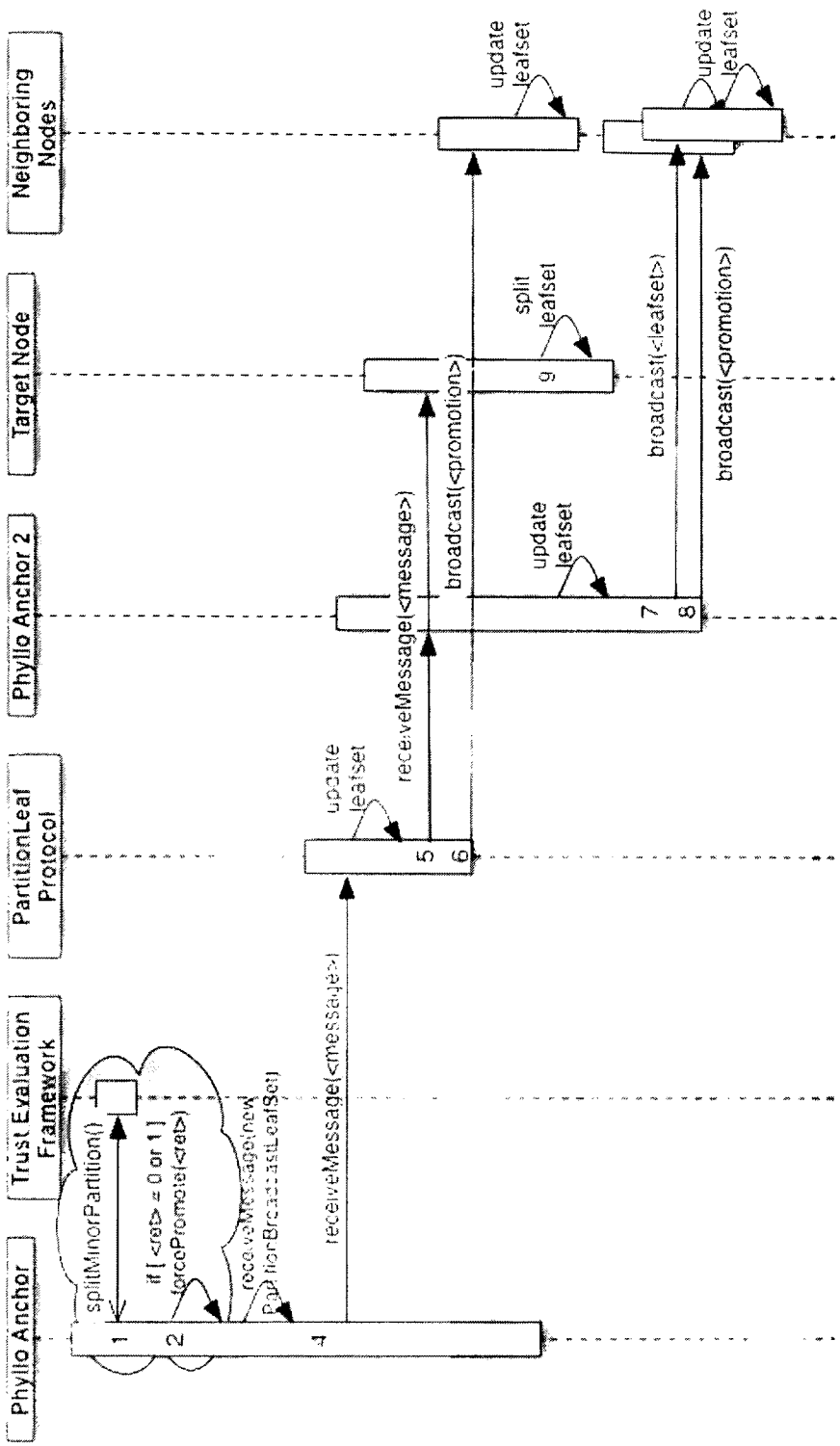

Figure 5.5: Sequence Diagram for the Force Promotion Protocol to split a partition

The demotion protocol's sequence diagram (Figure 5.6) at first glance, appears to be very similar to that of the promotion protocol. During the initial four step the demotion protocol is the exact same as the promotion protocol, with the "promotion" calls replaced with the demotion equivalent. The remainder of the steps are a bit more simplistic in that we do not have to worry about the second anchor node.

After the misbehaving node is identified, a `PartitionBroadcastLeafSet` message is created (step 4), and it is broadcast to the major partition (step 5), including the node targeted for demotion. Afterwards (steps 6, 8), the major partition nodes remove the demoted node from their *LeafSets*. It is done this way since the leaf set broadcast can only result in nodes being added. Therefore, we must have the demotion protocol explicitly handle the deletion of the malicious node. When the demoted node receives the message, in step 7, it must merge its leaf sets, if applicable. The resulting leaf set is then broadcast around the newly combined leaf sets. If the demoted node was not an anchor node, it will be moved into a new minor partition and the broadcast will only send to the two anchors for that partition.

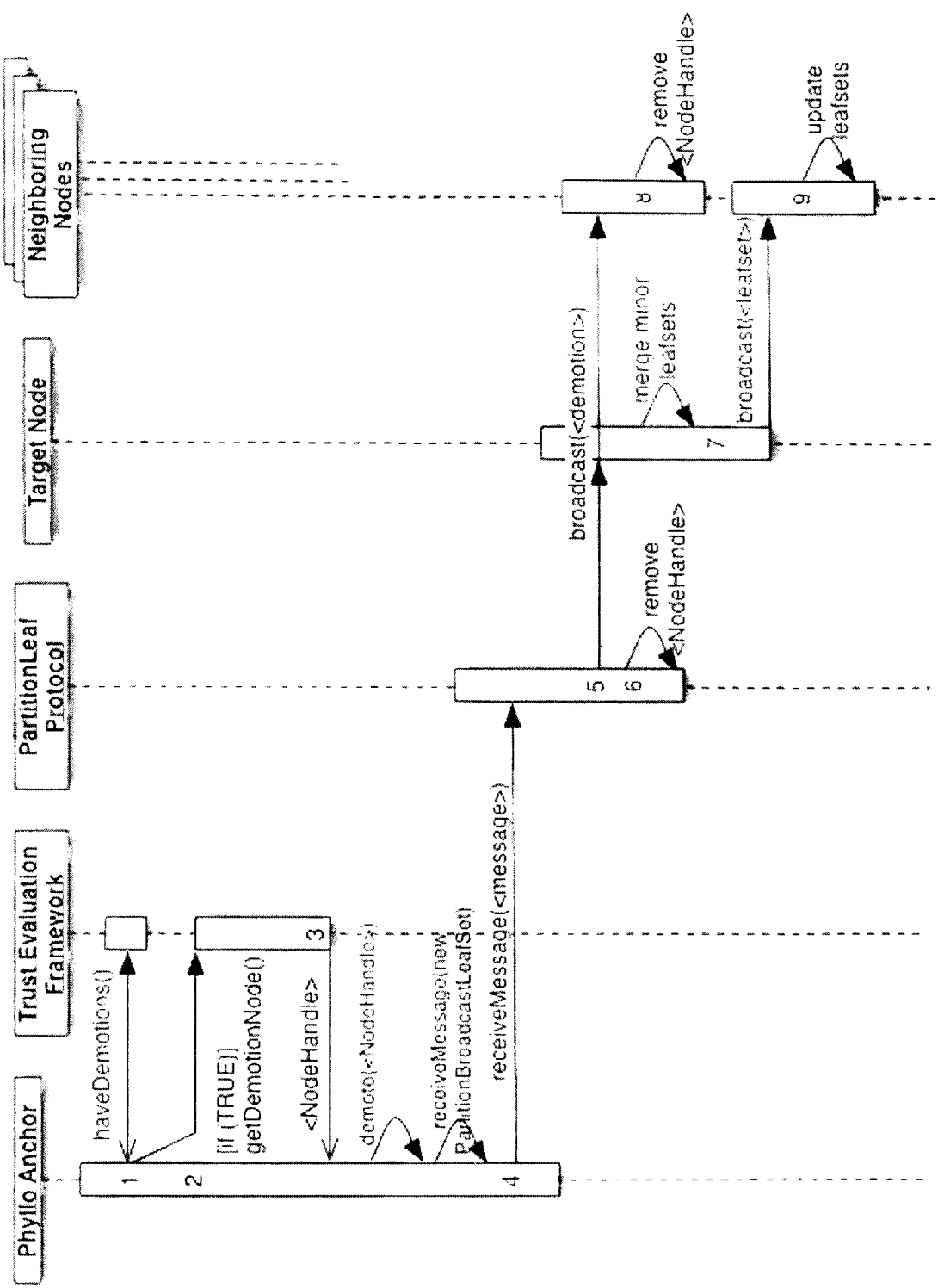

Figure 5.6: Sequence Diagram for a Demotion within Phyllo

Chapter 6

Results

> What we call the beginning is often the end. And to make an end is to make a beginning. The end is where we start from.
>
> — TS Eliot, *Four Quartets*[1]

After implementing Phyllo on the Pastry overlay network (Chapter 5), experiments were performed to measure the benefits and drawbacks of the framework. From the hierarchical structure, Phyllo should offer at least a minimal security boost over Pastry, but it would not be worth it if it came at the cost of significant overhead.

All of the experiments were performed using Pastry 1.4 on a Dual 2 GHz G5 PowerMac with 1 GB memory with Java 1.4.2. The size of the minor partition, R, was strictly bounded by the leaf set size, L. However, this is not a requirement for Phyllo but is used to force the promotion of nodes to ensure roughly equivalent minor partition sizes. The nodes were assigned random IDs in the ID space of 2^{128} Each of the experiments uses the following settings: ID base 16 ($b = 4$), leaf set size $|L| = 16$, maximum minor partition size $|R| = |L| = 16$, and routing table entries

[1] T.S. Eliot, "Four Quartets 4: Little Gidding," Poetry X, Ed. Jough Dempsey. July 2003. [Online]. Available: `http://poetry.poetryx.com/poems/758/`

$|M| = 16$. Additionally, the number of nodes, N, varies between 100 to 10,000 and will be specified on a per experiment basis.

The first tests were performed in order to obtain a comparative view of the security standing of Phyllo. Once assured of Phyllo's security potential, the additional overhead caused by the hierarchical structure was assessed.

6.1 Security Evaluation

The security-related attacks were done by performing some of the attacks outlined in Chapter 3: (1) ID Assignment Problem, (2) Message Forwarding Attack, and (3) Routing Table Maintenance Attack. The Lookup Attack was not specifically tested as it is a variation on the Message Forwarding Attack, and Pastry does not use the lookup protocol. Due to the nature of denial of service (DoS) attacks not targeting P2P flaws but rather its Internet foundation, they were also ignored in the testing phase.

6.1.1 ID Assignment & Routing Maintenance Results

Since there is no reliable way to test the ID Assignment and Routing Table Maintenance attacks individually and due to the fact that they are similarly perceived from the perspective of the overlay network, they were combined into a single test. This test involved a node properly joining the network. After the join was complete, the node would update network with false node IDs through transmitting bad routing updates.

Each trail was run with a static percentage of the total number of IDs being false. With a neighbor set of 16, maximum minor partition size of 16, and 5000 total nodes, 200,000 messages were sent from a randomly chosen source towards a

random number in the ID space. The tests were performed on both Pastry and Pastry with Phyllo for evaluation rates of $\alpha = 99\%$ and 90%. Data from these experiments is presented in Table 6.1.

The results show that Phyllo successfully mitigates this attack. When compared to an unmodified version of Pastry, Phyllo reduced the threat of routing attacks by more than a factor of 2 in some cases! Overall, as demonstrated in Figure 6.1, it is clear that Phyllo significantly improves the protection offered against the Sybil attack [26], and other ID Assignment or Routing Maintenance attack vectors. Phyllo was a success here, but can it offer protection against the Message Forwarding Attack?

6.1.2 Message Forwarding Attack Results

In this attack scenario, we try to duplicate the Byzantine Generals Problem. Some of the nodes on the network are malicious, i.e., once they receive a message, the integrity of that message is lost. Due to Phyllo's partitioning scheme and trust evaluation protocol, there should be less chance of one of these bad nodes needing to forward a message.

The overlay network is setup with a certain percentage of malicious nodes. Unlike the previous scenario, all of these nodes join the network byway of the join protocol. Whenever one of the malicious nodes receives a message that it should forward on to another node, the node drops the message instead. This is done because once the integrity of a message is lost, it can never be re-established; the number of malicious nodes, beyond the first, that touch a message is not important.

Table 6.2 shows the percentage of messages successfully delivered without be compromised. The data is from Pastry and Phyllo with $\alpha = 99\%$ or 90%, with 5000 messages exchanged. The same data is also visually represented in Figure 6.2. Here

Percentage False IDs	Pastry	Phyllo ($\alpha = 99\%$)	Phyllo ($\alpha = 90\%$)
0	0	0	0
0.1	0.26	0.10	0.12
0.2	0.47	0.17	0.23
0.3	0.65	0.27	0.32
0.4	0.77	0.36	0.42
0.5	0.85	0.44	0.53
0.6	0.92	0.54	0.61
0.7	0.95	0.62	0.67
0.8	0.98	0.72	0.77
0.9	1	0.8	0.83
1	1	1	1

Table 6.1: Percentage of messages routed through false node IDs ($b = 4, |R| = |L| = 16, |M| = 16, N = 5000$)

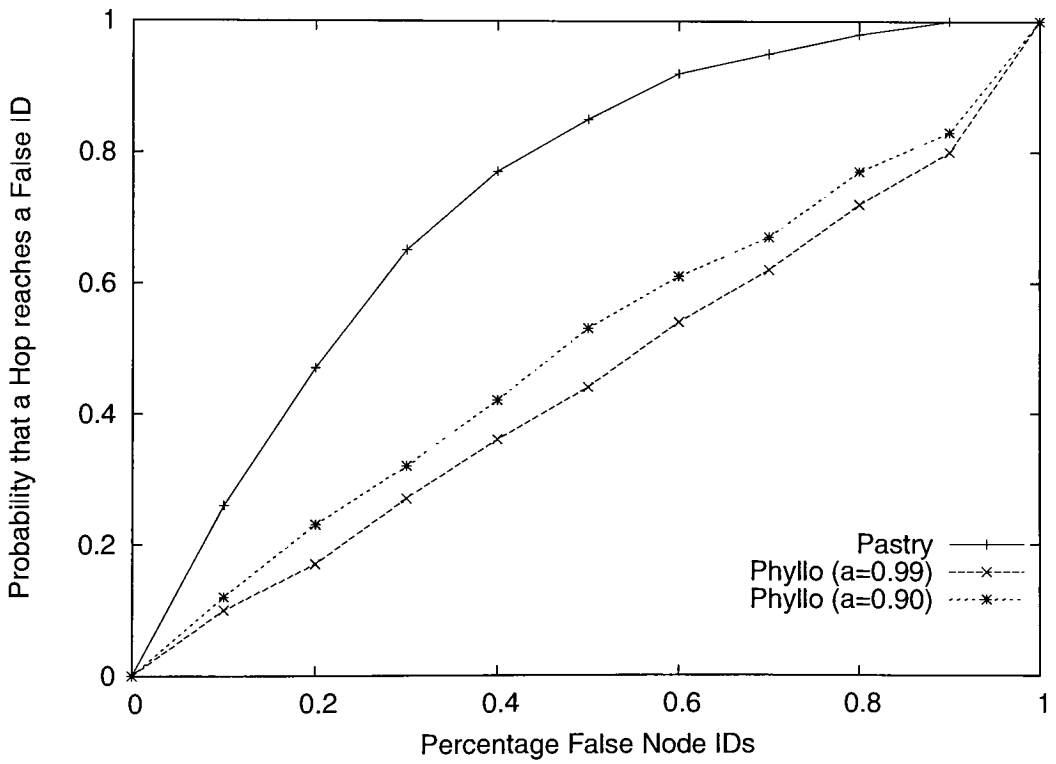

Figure 6.1: The Probability that a Message is Routed Through a False ID

it is evident that Pastry has a more polynomial correlation between the number of dropped or corrupted messages and the percentage of malicious nodes[2]. On the other hand, the same relation in Phyllo is mostly linear. Most importantly, Phyllo outperforms Pastry in every aspect by at least 10%!

For the average message (i.e., a message requiring 3 hops to route in Pastry or 3.75 hops in Phyllo[3]), Phyllo has a much better overall success rate (Figure 6.3). When only a relatively small percentage ($< 10\%$) of a network's nodes are malicious, Pastry tends to provide better reliability. However, Pastry's routing success rate plummets quickly as the percentage of malicious nodes increases. During this time, the same rate of success in Phyllo[4] remains fairly constant. Once malicious nodes make up 90% or more of any network, it becomes almost impossible to successfully route a message in either Phyllo or Pastry.

6.2 Networking Overhead

We have already proven that Phyllo successfully hardens an existing overlay network against some of their largest threats. Now, we must determine if Phyllo is worthy of implementation; i.e., does Phyllo add an acceptable amount of overhead? If the overhead is too high, the cost will outweigh the security benefits. As with all security-related issues, we are faced with the balance of security and efficiency.

The authors of Pastry [2] claim that it is capable of routing an application message between two randomly chosen nodes in no more than $\log_{2^b} n$ hops, where

[2]The chance for successful delivery in Pastry [2] under similar scenarios is $(1-p)^h$, where p is the percentage of malicious nodes and h is the total number of hops along a message's route.

[3]Average hop data taken from Table 6.3.

[4]$\left(1 - \frac{\alpha \times [\#\text{ of non-malicious major nodes}]}{[\#\text{ major nodes}]}\right)^h$

Percentage of Malicious Nodes	Pastry	Phyllo ($\alpha = 99\%$)	Phyllo ($\alpha = 90\%$)
0	1	1	1
0.10	0.74	0.90	0.88
0.20	0.54	0.80	0.75
0.30	0.36	0.70	0.62
0.40	0.24	0.60	0.55
0.50	0.15	0.50	0.46
0.60	0.07	0.40	0.37
0.70	0.04	0.30	0.25
0.80	0.014	0.21	0.18
0.90	0.005	0.11	0.10
1	0	0	0

Table 6.2: Percentage of messages delivered without being corrupted by a malicious node ($b = 4, |R| = |L| = 16, |M| = 16, N = 5000$)

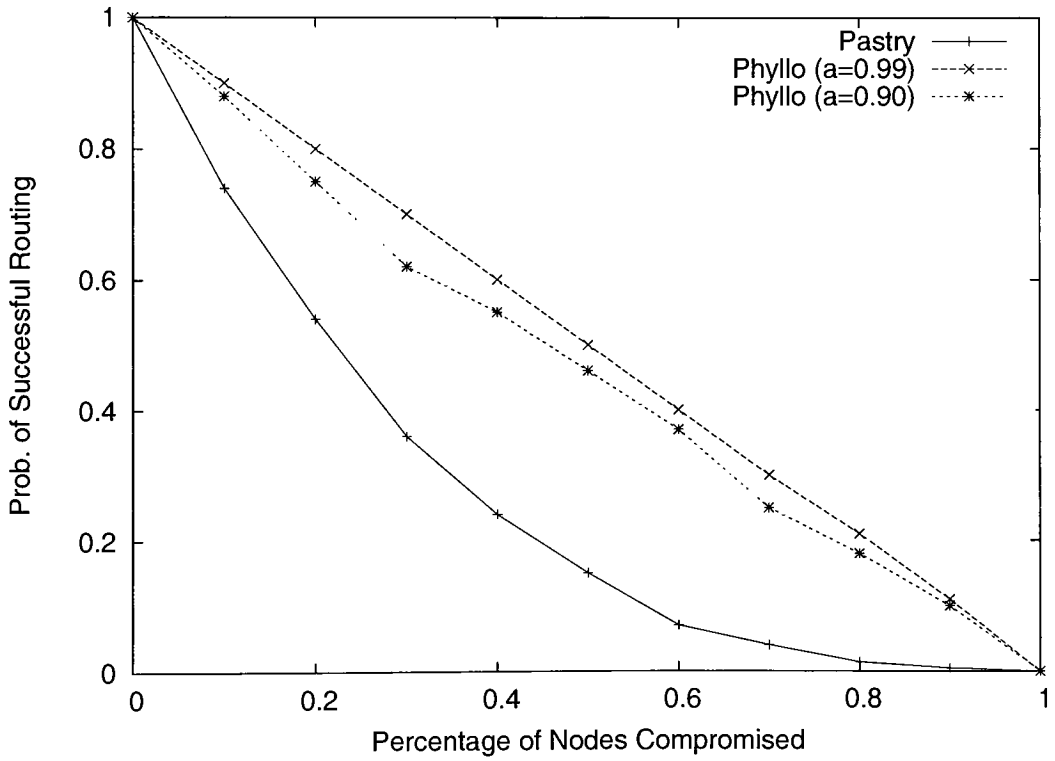

Figure 6.2: Chances for Successful Overlay Routing with $b = 4, |R| = |L| = 16, |M| = 16$, $N = 5000$, for various Phyllo α values.

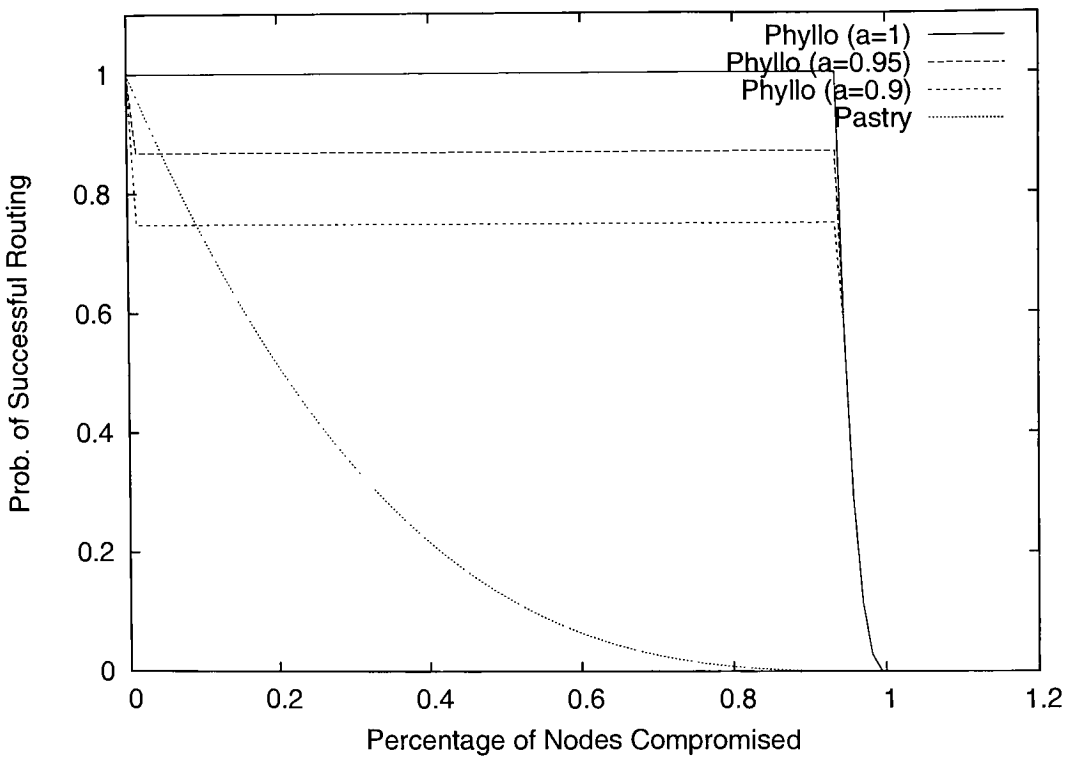

Figure 6.3: Comparison of Routing Success in a network consisting of 5000 nodes for a message requiring an average number of hops.

in this case, $b = 4$. Therefore, to be successful, Phyllo should be of the same order of magnitude, $\approx O(\log_{16} n)$ hops.

The experiment used to measure the average hop counts involved sending 200,000 messages from a random node towards a random destination. This was done on Pastry both with and without Phyllo, and was performed on overlay networks consisting of 100 to 10,000 nodes and $b = 4, |R| = |L| = 16, |M| = 16$. The hop count of each message was recorded, and the average (mean) of the data was used (Table 6.3).

Figure 6.4 shows the data from Table 6.3 plotted on a logarithmic axis for the network size. The experimental data gathered from Pastry is clearly $O(\log_{2^b} n)$. While the data from Phyllo appears to be of an equivalent order, it is not known. However, if we place an approximate upper boundary of $\log_{2^b} N+1$, Phyllo's average hop count would appear to agree.

Examining Phyllo's routing protocol, these results make logical sense. The modifications to Pastry's routing protocol (Section 4.2.2) call for an additional hop to be made when switching partitions. Therefore, in the worst case, an application message in Phyllo will have to be routed between two different minor partitions, whereby adding at most two additional hops. For each route in Pastry, the equivalent route in Phyllo will be either 0, 1, or 2 hops longer. When performed over a large number of routes, the average number of additional hops should be approximately 1.

We conclude that this is a reasonably small overhead to incur for the wealth of security benefits reaped because of Phyllo. Furthermore, when we analyzed the hop count data further, Phyllo actually provides a benefit to those networks that have to deal with unreliable network connection, such as an ad hoc network. While Phyllo provides a higher average hop count, it reduces the maximal hop count. Table 6.4 shows the maximum hop count recorded out of the 200,000 messages sent.

With a network size of 10,000 nodes, Phyllo drastically reduces the maximum

Network Size	Pastry	Phyllo
100	1.7	2.4
500	2.2	3.0
1000	2.5	3.25
2000	2.7	3.5
5000	3.0	3.75
8000	3.2	3.9
10000	3.33	4.15

Table 6.3: Average (mean) Hop Count for Messages Sent in Pastry and Phyllo ($b = 4, |R| = |L| = 16, |M| = 16$)

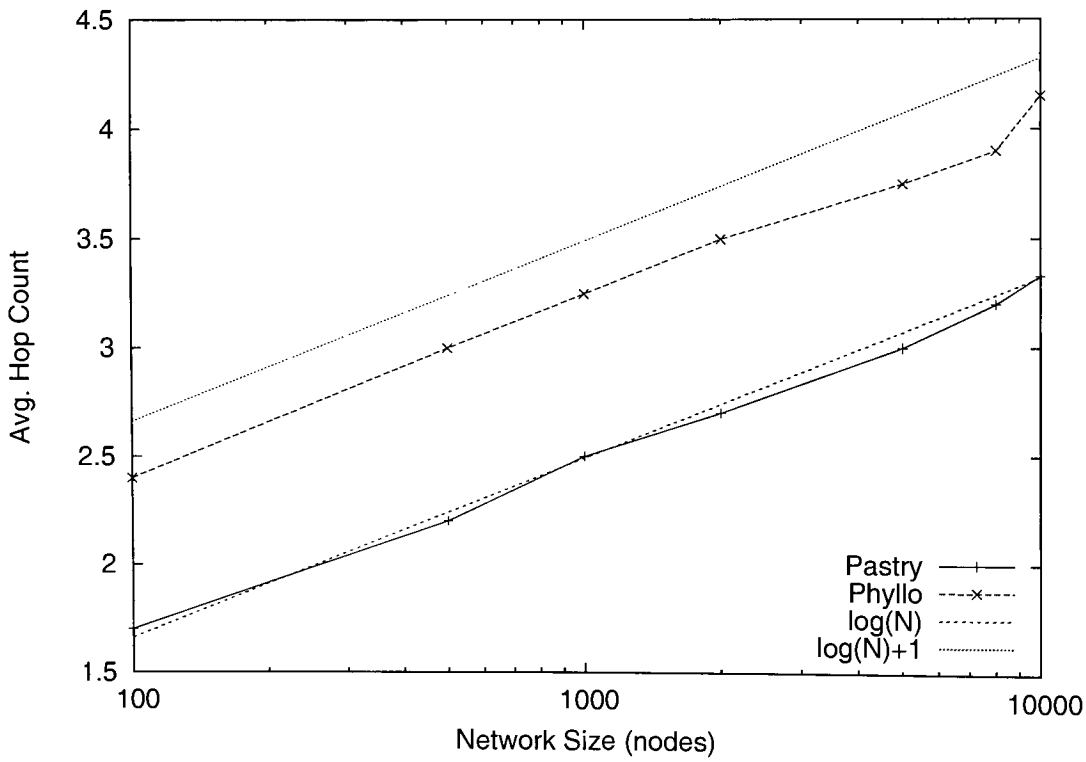

Figure 6.4: Average Hop Counts for Phyllo compared to Pastry Overlay Routing

Network Size	Pastry	Phyllo
100	3	4
500	3	5
1000	9	5
2000	9	6
5000	21	7
8000	22	8
10000	32	8

Table 6.4: Maximum Hop Count for Messages Sent in Pastry and Phyllo ($b = 4, |R| = |L| = 16, |M| = 16$)

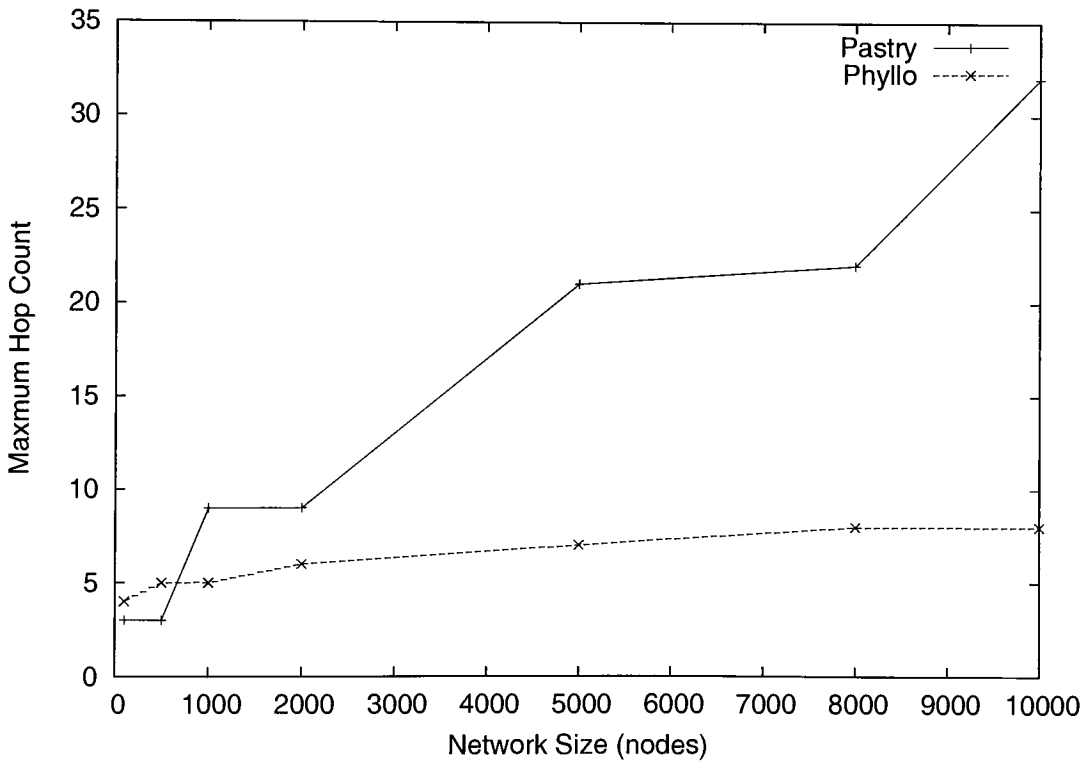

Figure 6.5: Maximum Number of Routing Hops in Pastry versus Phyllo

hop count required to route a message. While Pastry needs 32 hops, Phyllo reduces that by 75% to use only 8 hops! As the network grows, Pastry's maximal hop count shoots up from 3 to 32 hops. During that same period of growth, Phyllo uses only 4 more hops (ranging from 4 to 8 hops). At smaller network sizes, Phyllo uses only a couple of additional hops. This means that Phyllo is a better solution for networks where security is a major issue, or networks that have unreliable connections.

Chapter 7

Conclusion

> Few things are impossible to diligence and skill. Great works are performed not by strength, but perseverance.
>
> — Dr. Samuel Johnson

Overall, we feel that Phyllo was a success. It provides hardened security to architecturally weak overlay networks with only a slight network latency overhead. We recommend that all overlay networks that need heightened security or run over unreliable connections, implement Phyllo onto the basic overlay protocol.

However, the implementation of Phyllo on Pastry is just one example of the framework's power. The features such as adjustable partition size and depth allow for further customization, while the trust evaluation framework can be modified to fit any number of node control scenarios. The potential for Phyllo is even greater when you examine the wealth of overlay network candidates — from CAN to Chord and even Napster and BitTorrent. There is definitely a future for Phyllo.

The details for this future will be detailed in Section 7.1. Here, we will explain some of the possible research routes for Phyllo. Afterward, we will explore those projects with similar features and goals to Phyllo. This will provide a look into

related projects to allow for further research and comparison in the fields of peer-to-peer technologies and security.

7.1 Future Work

Thus far, we have only proven the basic concept of Phyllo, that the hierarchical layout and promotion/demotion protocols benefit the security in an overlay network. Phyllo was only implemented on a single overlay network and only the basic promotion/demotion protocol, along with the trust evaluation schema, functionality was evaluated. There is much room for further research.

Phyllo can be disected into three basic components: (1) hierarchical partitioning, (2) routing, and (3) promotion and demotion protocol, including the trust evaluation framework. In terms of the partitioning, only the most basic form was utilized. The hierarchy can be extended in both the number of levels and size. Increasing the maximum minor partition size (up to infinity), allows for a more secure major partition. At the same time, it places a burden on the anchor nodes and may hamper the routing efficiency. One possible research question is to determine the most efficient (i.e., the best security to overhead/performance ratio) minor partition size.

A related question would concern the best security to number of hierarchical layers trade off. If the number of partition layers were to increase beyond two, to what extent would there be a benefit to security? Doing this would definitely increase the overall hop counts, but is that sacrifice worth it? There is most likely some optimal number of layers and ranges of partition sizes for networks with concerns such as security, connectivity, bandwidth, resources (memory or computational), etc.

Each of these decisions impacts the network routing. Another variable in the search for the best Phyllo implementation is the underlying routing algorithm. This time around, we chose to inherit Pastry's network structure and routing algorithm. Would a routing algorithm based on Euclidean distances, such as CAN, be better? What about a network similar to Chord which supports lookups, and will send the messages directly to the lookup's result?

The final research area, and possibly the most interesting, involves the promotion and demotion protocols. Phyllo is possibly the only network to feature the possibility for nodes to dynamically move between partitions. With this advantage comes a number of unanswered questions: when is the right time to promote? demote? Is there any way which a malicious node can exploit the promotion or demotion protocol? Should a misbehaving node be kicked from the network if it misbehaves and cannot be further demoted?

Acting in support of these protocols, the trust evaluation framework will also play a role in maximizing an overlay network's capability. In the future, we would like to explore the various metrics that we could use to evaluate a node. The notion of the framework could be extended to allow for groups to decide a node's fate; maybe something similar to a majority vote is needed to take action against a node. We could also experiment with the threshold settings used as decision makers in the protocols. Generally, we would like to know, what is the best trust evaluation framework for a given scenario?

Additionally, some of current trust evaluation criteria, such as ensuring the proper forwarding of messages, relies on information that can be manipulated. Nodes can change the sending address to appear to have come from elsewhere; there is no validation performed. While this is probably specific to the Pastry implementation, it does pose potential security problems and should be fixed.

We will also attempt to evaluate Phyllo in a larger environment. The data presented in Chapter 6 was gathered from locally performed experiments with a maximum of 10,000 nodes. To more accurately reflect the product's end use, it is necessary to test with a larger population of nodes that are spread across a more realistic, network environment. The network research environment provided via collaborations such as planetlab [32], may provide the proper testbed.

Bibliography

[1] W. Heinbockel and M. Kwon, "Phyllo: A Peer-to-Peer Overlay Security Framework," in *First Workshop on Secure Network Protocols (NPSec)*, To Appear in November 2005.

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of ACM/IFIP/USENIX Middleware*, Heidelberg, Germany, November 2001, pp. 329–350.

[3] T. Sundsted, "The practice of peer-to-peer computing: Introduction and history." [Online]. Available: http://www-106.ibm.com/developerworks/java/library/j-p2p/

[4] G. Spafford, "Usenet Software: History and Sources," December 1999. [Online]. Available: ftp://rtfm.mit.edu/pub/usenet-by-group/news.admin.misc/Usenet_Software:%_History_and_Sources

[5] R. Bush, "FidoNet: Technology, Tools, and History," *Commun. ACM*, vol. 36, no. 8, pp. 31–35, 1993.

[6] I. Clarke, O. Sandberg, B. Wiley, and T. W Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," in *Workshop on Design Issues in Anonymity and Unobservability.* Berkeley, California: ICSI, July 2000, pp. 311–320.

[7] J. Tyson, "How the Old Napster Worked," HowStuffWorks, Inc., 2005. [Online]. Available: http://www.howstuffworks.com/napster.htm

[8] J.-M. Wong, "Gene Kan: How Gnutella Happened," February 2001. [Online]. Available: http://news.dmusic.com/print/3641

[9] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *Proceedings of ACM SOSP*, 2001, pp. 188–201.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in a cooperative

environment," in *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[11] A. Rowstron, A.-M. Kermarrec, M. Castro, and P Druschel, "Scribe: The design of a large-scale event notification infrastructure," in *Proceedings of NGC*, ser. Lecture Notes in Computer Science, J. Crowcroft and M. Hofmann, Eds., vol. 2233, 2001, pp. 30–43.

[12] Skype Technologies S.A., "Skype The whole world can talk for free." [Online]. Available: http://www.skype.com/

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable peer-to-peer lookup service for Internet applications," in *Proceedings of ACM SIGCOMM*, March 2001, pp. 149–160.

[14] "The Gnutella protocol specification." [Online]. Available: http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

[15] J. Brown, "The Gnutella paradox," Salon.com, September 2000. [Online]. Available: http://www.salon.com/tech/feature/2000/09/29/gnutellaparadox/

[16] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," in *Proceedings of USENIX OSDI.* ACM Press, 2002.

[17] E. Sit and R. Morris, "Security Consideration for Peer-to-Peer Distributed Hash Tables," in *Proceedings of IPTPS*, Cambridge, MA, March 2002.

[18] L. Garcés-Erice, E. Biersack, P. Felber, K. Ross, and G. Urvoy-Keller, "Hierarchical Peer-to-peer Systems," in *Proceedings of IASTED PDCS.* Springer-Verlag, LNCS, 2003.

[19] Z. Xu, R. Min, and Y. Hu, "HIERAS: a DHT based hierarchical P2P routing algorithm," in *Proceedings of IEEE ICPP*, October 2003, pp. 187–194.

[20] R. Hsiao and S.-D. Wang, "Jelly: a dynamic hierarchical P2P overlay network with load balance and locality," in *Proceedings of IEEE ICDCS*, March 2004, pp. 534–540.

[21] K. Aberer and Z. Despotovic, "Managing Trust in a Peer-2-Peer Information System," in *Proceedings of ACM CIKM.* ACM Press, 2001, pp. 310–317.

[22] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks," in *Proceedings of ACM WWW*, 2003, pp. 640–651.

[23] M. Waldman and D. Mazières, "Tangler: a censorship-resistant publishing system based on document entanglements," in *Proceedings of ACM CCS*. ACM Press, 2001, pp. 126–135.

[24] E. Friedman and P. Resnick, "The Social Cost of Cheap Pseudonyms," *Journal of Economics and Management Strategy*, vol. 10, no. 2, pp. 173–199, June 2001.

[25] A. D. Keromytis, V Misra, and D. Rubenstein, "SOS: Secure Overlay Services," in *Proceedings of ACM SIGCOMM*, 2002, pp. 61–72.

[26] J. R. Douceur, "The Sybil Attack," in *Proceedings of the IPTPS*, Cambridge, MA, March 2002.

[27] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM TOPLAS*, vol. 4, no. 3, pp. 382–401, 1982.

[28] C. Cachin, K. Kursawe, and V Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography," in *Proceedings of the Symposium on Principles of Distributed Computing*, 2000, pp. 123–132.

[29] D. Wallach, "A Survey of Peer-to-Peer Security Issues," in *Proceedings of ACM ISSS*, 2002.

[30] S. J. Nielson, S. A. Crosby, and D. S. Wallach, "A Taxonomy of Rational Attacks," in *Proceedings of IPTPS*, February 2005.

[31] A. T. Fiore, S. L. Tiernan, and M. A. Smith, "Observed behavior and perceived value of authors in usenet newsgroups: bridging the gap," in *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, 2002, pp. 323–330.

[32] PlanetLab Consortium, "Planetlab." [Online]. Available: http://www.planet-lab.org/

[33] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," Massachusetts Institute of Technology, Tech. Rep., January 2002. [Online]. Available: http://pdos.lcs.mit.edu/chord/

[34] N. Saxena, G. Tsudik, and J. H. Yi, "Admission Control in Peer-to-Peer: Design and Performance Evaluation," in *Proceedings of ACM SASN*. ACM Press, 2003, pp. 104–113.

[35] D. Clark, K. Sollins, J. Wroclawski, and T. Faber, "Addressing Reality: An Architectual Response to Real-World Demands on the Evolving Internet," in *Proceedings of the ACM SIGCOMM 2003 FDNA Workshop*, Karlsruhe, Germany, August 2003.

Appendix A

User's Guide

A.1 Installation

The files referred to within this document can be found on the accompanying CD-ROM, via the Phyllo project website (`http://www.cs.rit.edu/~wjh3710/thesis`), or through the provided URLs.

The source code of Phyllo is written in Java and tested using Java 1.4.2. Before any of this code can be run or compiled, Java SDK version 1.4.2 or later must be installed on the target system. The latest versions of the Java SDK can be found on Sun Microsystems's Java website: `http://java.sun.com`.

This implementation of Phyllo requires Pastry, which can be obtained off of the official Pastry website: `http://freepastry.rice.edu/`. Phyllo was designed and tested on Pastry 1.4.0 and due to modifications made to the messaging protocols in the later releases (Pastry 1.4.1 at the time of publication), Phyllo is not directly compatible with those releases. Additionally, in order to get Phyllo to work, a patch must be applied to `rice/pastry/leafset/LeafSet.java` to change the scope of the `baseId` variable from *private* to *protected*. This is necessary so that Phyllo can

extend this class.

This process has already been completed on the Pastry files on the accompanying CD. Additionally, in the patches directory, there is a patch file and script to apply the patch to the `LeafSet.java` file. After this process, the code must be recompiled. Instructions for installing and running FreePastry are located in `FreePastry-1.4-source/docs/README.html`.

Pre-patched versions of the source code and Java archive (jar) are available on the accompanying CD in `FreePastry-1.4-source/src` or `FreePastry-1.4-modified.jar` in the main directory. Accompanying distribution files are also included in the `FreePastry-1.4-source` directory, as well as the Pastry JavaDocs provided by Rice University, in `FreePastry-1.4-source/docs/javadocs/`.

The Phyllo source code can be found in `Phyllo/src`, while an already compiled archive, `Phyllo.jar` is located in the main directory. A complete set of JavaDocs can also be found in `Phyllo/docs`. A copy of this documentation is located in `Instructions.html` on the CD or at `http://www.cs.rit.edu/~wjh3710/thesis/Instructions.html`.

A.2 Running Phyllo

The Phyllo implementation contains three test classes. These main classes are located in the `rit.secure.pastry.testing` package and are named `SecHelloWorld`, `TestRouteTableDurability`, and `GatherMessageData`.

In order to run any of the tests, the Java executable must be informed of the locations of the Phyllo and Pastry classes to use. This can be done by declaring the `$CLASSPATH` environment variable or using the `-cp` flag with the `java` executable. Either way, Java needs to know the path for the root package of the compiled *.class*

files or full path of the JAR archives. For example, to execute the `SecHelloWorld` program from the root of the CD, one would enter:

```
bash% java -cp ./Phyllo.jar:./FreePastry-1.4-modified.jar\
      rit.secure.pastry.testing.SecHelloWorld
```

The `SecHelloWorld` can be replaced by any of the other two test files' names to run them. Additionally, there are arguments that accompany each program, which can be seen with the `-help` flag or described in that class's JavaDoc. Here is a listing and description of the most important options:

-msgs m The number of messages, m, to send after the nodes have been created.

-nodes n The number of standard nodes, n, to create.

-malicious o The number of malicious nodes, o, to create.

-eval a The α value, a, used to simulate the efficiency of the Trust Evaluation Framework.

-phyllo Run the test using the Phyllo framework with Pastry.

-pastry Run the test using only Pastry.

A.2.1 Producing the Results

This section will explain the process and scripts used to generate the results presented in Chapter 6 of my thesis paper.

The files discussed within this section are located in the `Experiments/Scripts/` directory on the accompanying CD-ROM. Here is a listing of the directory contents:

data_parsers – Directory containing scripts to parse the program output

data_parsers/success_count.awk – Counts the number of successfully routed messages versus the total number of messages sent with `run_route_test.sh`.
Usage: $awk - f data\_parsers/success\_count.awk < data\_file >$

data_parsers/summarize.awk – Collects the hop count data output from the `run_hop_test.sh` script.
Usage: $awk - f data\_parsers/summarize.awk < data\_file >$

data_parsers/sybil_summary.awk – Determines how often a message was received by an invalid node. Used with the `run_fwd_test.sh` test.
Usage: $awk - f data\_parsers/sybil\_summary.awk < data\_file >$

README.txt – Explains the scripts usage and other notes.

run_fwd_test.sh – Run one instance of the simulation with malicious applications to simulate false nodes (Sybil Attack)
Usage: $./run\_fwd\_test.sh < output\_file >< \%malicious > [phyllo|pastry]$

run_hop_test.sh – Establish a network and send test messages to gather each message's final hop count.
Usage: $./run\_hop\_test.sh < output\_file >< \#nodes >< \#msgs > [phyllo|pastry]$

run_repeated.sh – Run the `run_route_test.sh` script for networks containing 0-90% (in 10% increments) of malicious nodes.
Usage: $./run\_repeated.sh < alpha > [phyllo|pastry]$

run_route_test.sh – Run a test over a network with a specified concentration of malicious nodes. Each malicious node will only send or receive a message. If a message should be forwarded, a malicious node will drop that message.
Usage: $./run\_route\_test.sh < output\_file >< \%malicious >$
$< alpha > [phyllo|pastry]$

Each of the bash shell scripts take a couple of parameters. Most should be self-explanatory, but this is just a reminder that *alpha* refers to the α efficiency metric for the Trust Evaluation Framework and is simulated in these experiments. Also, the user may specify either "phyllo" or "pastry" as an input to each script. This tells the program whether or not to use the Phyllo framework in addition to the standard Pastry network (identical functionality to the `-phyllo` and `-pastry` flags in the previous section).

The results of running these scripts can be found in `Experiments/Results/` This is the same data and graphs that were presented previously in this thesis. Since the Java programs outlined above make use of a random number generator to assign IDs, and determine message sources and destinations, individual results are guaranteed to vary for larger network sizes.

A.2.2 Writing Phyllo Applications

These experimental programs provide a nice layout and starting point to enable developers to begin writing more functional Phyllo applications.

Pastry (and Phyllo) require two different "application" types in order to create a single P2P application. The first is a node application. This application is part of the node and handles all application-specific protocols, such as searching, file transfer, user messaging — any application message. Multiple node applications can be run on the same overlay because they are identified using an address unique to each application type. Examples of node applications are `rit.secure.pastry.testing.GatherDataApp` and `rit.secure.pastry.testing.RoutingTableTestApp`.

The other application type is the main class (containing the *main* method) and is used to initialize the node and node application. This initialization process involves assigning the node an ID and then starting the bootstrapping, or join, protocol. "Main applications" such as `GatherMessageData` and `TestRouteTableDurability` demonstrate the minimal responsibilities that this application type provides.

The last major piece necessary to write a P2P application that uses Phyllo, is the message. Pastry passes instances of the `rice.pastry.messaging.Message` object between the various overlay nodes. The developer is responsible for creating a message specific to the node application it is to be supporting. All necessary application-level data should be encapsulated inside of a message instance. `rit.secure.pastry.testing.TraceMessage` is one message example that can be used to track message hop counts.

Appendix B

Phyllo JavaDocs

The following list of classes, and their associated JavaDocs, can be found within the distribution of the Phyllo framework:

- `rit.secure.pastry`
  - ▷ `SecPastryNode`
- `rit.secure.pastry.direct`
  - ▷ `DirectSecPastryNode`
  - ▷ `DirectSecPastryNodeFactory`
- `rit.secure.pastry.direct.malicious`
  - ▷ `MaliciousDirectPastryNode`
  - ▷ `MaliciousPastryNodeFactory`
  - ▷ `MaliciousSphereNetwork`
- `rit.secure.pastry.partition`
  - ▷ `PartitionBroadcastLeafSet`
  - ▷ `PartitionLeafSet`
- `rit.secure.pastry.promotion`
  - ▷ `DemotionMessage`
  - ▷ `ForcedPromotionEvaluation`

- ▷ PromotionEvalutation
- ▷ PromotionMessage
- ▷ PromotionProtocolAddress

- rit.secure.pastry.standard
  - ▷ PartitionJoinProtocol
  - ▷ PartitionLeafSetProtocol
  - ▷ PartitionPromotionProtocol
  - ▷ PartitionRouteSetProtocol
  - ▷ PartitionRouter

- rit.secure.pastry.standard.malicious
  - ▷ MaliciousLeafSetProtocol

- rit.secure.pastry.testing
  - ▷ BasicEvaluation
  - ▷ EvaluationData
  - ▷ GatherDataApp
  - ▷ GatherDataApp.DataAddress
  - ▷ GatherMessageData
  - ▷ MinorPartitionEnforcement
  - ▷ RoutingTableTestApp
  - ▷ RoutingTableTestApp.DataAddress
  - ▷ SecHelloWorld
  - ▷ TestRouteTableDurability
  - ▷ TraceMessage